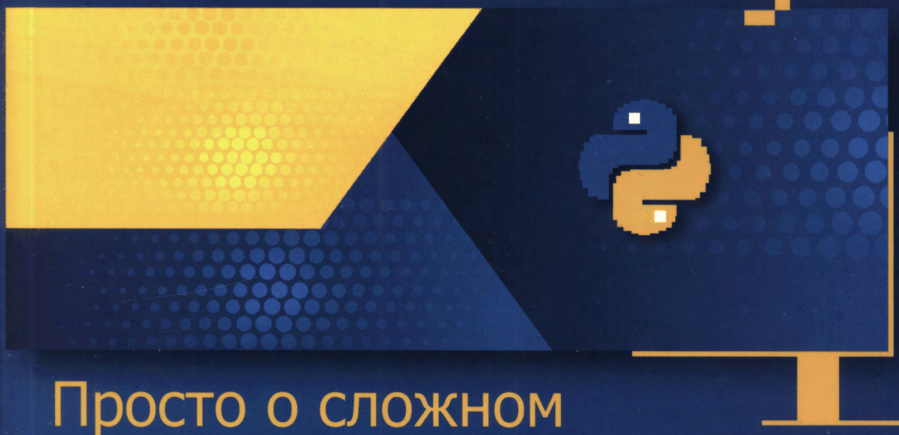
Начнем. Python



Просто о сложном

ИВАНОВ С. С.

НиТ
ИЗДАТЕЛЬСТВО

Иванов С. С.

НАЧНЕМ PYTHON

ПРОСТО О СЛОЖНОМ

"Издательство Наука и Техника"

Санкт-Петербург

УДК 004.42
ББК 32.973

Иванов С. С.

Начнем.Python. Просто о сложном — СПб.: Издательство Наука и Техника, 2023. — 368 с., ил.

ISBN 978-5-907592-23-0

Данная книга поможет вам в изучении языка программирования **Python**.

Многоцелевая направленность Python позволяет решать самые разные задачи: математические, физические, лингвистические, бухгалтерские, экономические, заниматься созданием игр(!), работой в Интернете и многое другое.

Отличием книги является легкий, понятный и даже где-то юмористический подход автора к программированию на Python, благодаря чему вы будете учиться писать программы, ошибаться вместе с автором, исправлять ошибки и любоваться на готовые работоспособные программы...

В книге рассмотрены все ключевые аспекты Python: переменные, операторы, логические выражения, циклы, комментарии, функции, рекурсия, строки, кортежи, словари и множества. В каждой главе теоретическую часть дополняют многочисленные практические примеры и задания для самостоятельного решения (впрочем, для самых нетерпеливых в конце книги приводятся правильные ответы).

Книга предназначена для широкого круга читателей, не требует навыков в программировании и будет полезна всем, кто хочет познакомиться с Python.

ISBN 978-5-907592-23-0

Контактные телефоны издательства:

(812) 412 70 26

Официальный сайт: www.nit.com.ru

Содержание

Введение.....11

Глава 1. ПЕРЕМЕННЫЕ.....19

Глава 2. АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ.....25

Глава 3. Первая программа.....33

Глава 4. Ещё операторы.....43

Глава 5. УСЛОВНЫЕ ОПЕРАТОРЫ.....63

Глава 6. ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ.....71

Глава 7. ЦИКЛЫ.....79

Глава 8. КОММЕНТАРИИ.....95

Глава 9. ОБОЛОЧКА IDLE.....99

Глава 10. ФУНКЦИИ.....105

Глава 11. МАТЕМАТИЧЕСКИЕ ФУНКЦИИ.....121

Глава 12. РЕКУРСИЯ.....133

Глава 13. СТРОКИ.....139

Глава 14. СПИСКИ.....155

Глава 15. КОРТЕЖИ.....169

Глава 16. СЛОВАРИ.....177

Глава 17. МНОЖЕСТВО.....185

Глава 18. Что-то пошло не так.....197

Глава 19. Извне.....207

Глава 20. Ну-ка все вместе.....219

Глава 21. Время поиграть.....235

ОТВЕТЫ НА УПРАЖНЕНИЯ.....241

УПРАЖНЕНИЕ 0.1.....242

УПРАЖНЕНИЕ 1.1.....242

УПРАЖНЕНИЕ 2.1.....242

УПРАЖНЕНИЕ 2.2.....243

УПРАЖНЕНИЕ 3.0.....244

УПРАЖНЕНИЕ 3.1.....244

УПРАЖНЕНИЕ 3.2.....246

УПРАЖНЕНИЕ 3.3.....247

УПРАЖНЕНИЕ 3.4.....248

УПРАЖНЕНИЕ 4.1.....250

УПРАЖНЕНИЕ 4.2.....250

УПРАЖНЕНИЕ 4.3.....252

УПРАЖНЕНИЕ 4.4.....253

УПРАЖНЕНИЕ 4.5.....253

УПРАЖНЕНИЕ 4.6.....254

УПРАЖНЕНИЕ 4.7.....254

УПРАЖНЕНИЕ 5.1.....255

УПРАЖНЕНИЕ 5.2.....256

УПРАЖНЕНИЕ 5.3.....257

УПРАЖНЕНИЕ 5.4.....257

УПРАЖНЕНИЕ 5.5 258
УПРАЖНЕНИЕ 6.1 260
УПРАЖНЕНИЕ 6.2 261
УПРАЖНЕНИЕ 6.3. 262
УПРАЖНЕНИЕ 6.4 262
УПРАЖНЕНИЕ 6.5 263
УПРАЖНЕНИЕ 6.6 263
УПРАЖНЕНИЕ 6.7 263
УПРАЖНЕНИЕ 7.1 264
УПРАЖНЕНИЕ 7.2 264
УПРАЖНЕНИЕ 7.3 265
УПРАЖНЕНИЕ 7.4 265
УПРАЖНЕНИЕ 7.5 265
УПРАЖНЕНИЕ 7.6 266
УПРАЖНЕНИЕ 7.7 267
УПРАЖНЕНИЕ 7.8 269
УПРАЖНЕНИЕ 7.9 271
УПРАЖНЕНИЕ 10.1 273
УПРАЖНЕНИЕ 10.2 273
УПРАЖНЕНИЕ 10.3 274
УПРАЖНЕНИЕ 10.3.1 275
УПРАЖНЕНИЕ 10.4 275
УПРАЖНЕНИЕ 10.5 276
УПРАЖНЕНИЕ 10.6 277

УПРАЖНЕНИЕ 11.1 278
УПРАЖНЕНИЕ 11.2 279
УПРАЖНЕНИЕ 11.3 285
УПРАЖНЕНИЕ 11.4 285
УПРАЖНЕНИЕ 12.1 286
УПРАЖНЕНИЕ 12.2 287
УПРАЖНЕНИЕ 12.3 287
УПРАЖНЕНИЕ 13.1 288
УПРАЖНЕНИЕ 13.2 288
УПРАЖНЕНИЕ 13.3 291
УПРАЖНЕНИЕ 13.3.1 291
УПРАЖНЕНИЕ 13.4 292
УПРАЖНЕНИЕ 13.5 292
УПРАЖНЕНИЕ 13.6 293
УПРАЖНЕНИЕ 14.1 294
УПРАЖНЕНИЕ 14.2 294
УПРАЖНЕНИЕ 14.3 294
УПРАЖНЕНИЕ 14.4 295
УПРАЖНЕНИЕ 14.5 296
УПРАЖНЕНИЕ 14.6 296
УПРАЖНЕНИЕ 14.7 298
УПРАЖНЕНИЕ 14.8 300
УПРАЖНЕНИЕ 15.1 302
УПРАЖНЕНИЕ 15.2 303

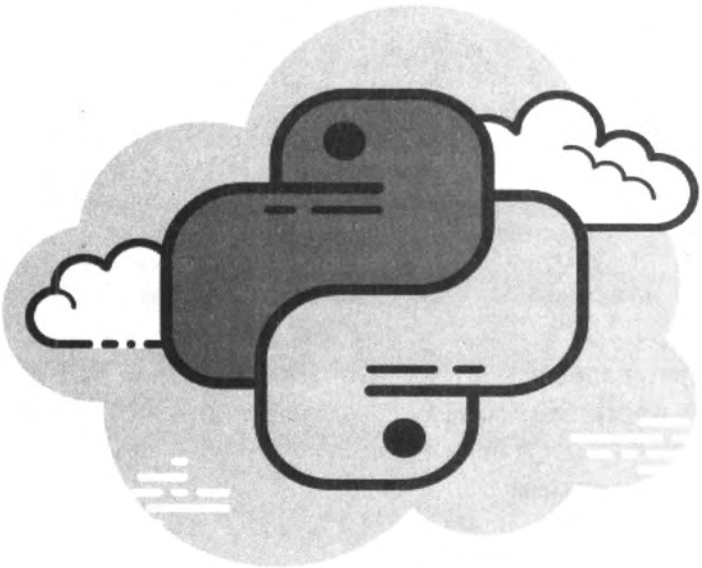
УПРАЖНЕНИЕ 15.3 305
УПРАЖНЕНИЕ 15.4 308
УПРАЖНЕНИЕ 16.1 309
УПРАЖНЕНИЕ 16.2 310
УПРАЖНЕНИЕ 16.3 311
УПРАЖНЕНИЕ 17.1 312
УПРАЖНЕНИЕ 17.2 313
УПРАЖНЕНИЕ 17.3 317
УПРАЖНЕНИЕ 17.4 318
УПРАЖНЕНИЕ 18.1 320
УПРАЖНЕНИЕ 18.2 321
УПРАЖНЕНИЕ 19.1 322
УПРАЖНЕНИЕ 19.2.1 323
УПРАЖНЕНИЕ 19.2.2 324
УПРАЖНЕНИЕ 19.2.3 325
УПРАЖНЕНИЕ 19.3 326
УПРАЖНЕНИЕ 20.1 330
УПРАЖНЕНИЕ 20.2 331
УПРАЖНЕНИЕ 20.3 332
УПРАЖНЕНИЕ 20.4.1 333
УПРАЖНЕНИЕ 20.4.2 334
УПРАЖНЕНИЕ 21.1 335
УПРАЖНЕНИЕ 21.2 336
УПРАЖНЕНИЕ 21.3 336

Приложение 1. Основные методы обработки строк.....345

Приложение 2. Основные методы обработки списков.....355

Приложение 3. Основные методы обработки словарей.....361

ВВЕДЕНИЕ



Автор в шутку называет эту книгу полуучебником. Что такое? Почему полуучебник? Как это прикажете понимать? Это только половина учебника? А где вторая?

Да нет, конечно... Книга, которую вы держите в руках – как и всякий учебник, содержит в себе массу упражнений, в которых вы будете учиться писать программы на чудесном языке Питон, ошибаться вместе с автором, исправлять ошибки и любоваться на готовые работоспособные программы... А вот в отличие от учебника книга написана нормальным, а не скучным академическим языком, весело и даже занимательно. Поэтому и получился у нас полуучебник – не совсем учебник, но и вовсе даже не роман.

Итак, что за зверь такой – питон?

Ну, начнём с того, что питон – это не зверь, а пресмыкающееся (автор, как и всякий учитель, немного зануда).

А вот Python (далее просто Питон) — **многоцелевой высокоуровневый язык программирования**. Его название и правда переводится как «питон» – змея, а произносится как ['paɪθən] (в связи с тем, что русскоязычным людям очень плохо даются звуки θ и ə, мы не будем умничать и будем называть этот язык просто Питон, но уважительно – с большой буквы). Но! Назван он не в честь змеи (обидно, да?), а в честь британской комик-группы, а точнее, популярного телешоу (!) «Летающий цирк Монти Пайтона».

Что значит многоцелевой? Хороший вопрос, как сейчас модно выражаться! Это значит, что с помощью Питона можно решать самые разные задачи: математические, физические, лингвистические, бухгалтерские, экономические, заниматься созданием игр(!), работой в Интернете...

А что такое высокоуровневый? О!... Это долгая история...

Начнём с того, что компьютер – это набор огромного количества транзисторов, тиристоров, резисторов, процессоров... какие ещё слова с окончанием на «–ор» вы знаете? Там, внутри, почти всё это есть... Так вот, вся эта груда компонентов совершенно не знает, чего вы от неё хотите! Для объяснения этим железякам что надо делать, и служит **Программное Обеспечение**. Одни программы зажигают экран и выводят на него иконки, другие – определяют, что я нажал на клавиатуре, третьи – выводят данные на принтер и музыкальные колонки. А вам в это время надо срочно решить квадратное (а то и совсем не квадратное) уравнение. Или сосчитать количество букв в поэме «Полтава». Или сосчитать, сколько дней вы уже прожили на свете и когда отмечать свой МИЛЛИОННЫЙ день рождения. Все эти задачи мы с вами скоро решим с помощью Питона.

Компьютер, как вы, надеюсь, знаете, понимает только свой внутренний язык, состоящий из последовательностей нулей и единиц. Поэтому первоначально программы выглядели примерно так:

```
11101000100011101000110111000010001110101111010100110111011000
01000111110111010111100
```

(и на самом деле в процессоре компьютера они и сейчас так выглядят).

Однако очень быстро первые программисты, измученные головной болью, придумали, как избавиться от этого кошмара. Дело в том, что этот поток нулей и единиц на самом деле компьютер воспринимает кусками определённого размера, а каждый кусок означает для компьютера определённую команду, например: возьми данные из ячейки 1010100010101101 внешней (довольно медлительной) памяти и загрузи их во второй регистр оперативной (то есть очень быстро исполняемой) памяти, а теперь к первому регистру добавь содержимое второго регистра, а результат помести назад, в первый регистр.

Такого рода команды уже можно записать более-менее понятным хорошо подготовленному человеку языком, именуемым **Ассемблер** (от английского Assemble – собирать).

Получается примерно вот так:

```
LDA, R2
RST 16
LDA, R1
SUMR1, R2
```

Дальше компьютер сам, с помощью несложной программы, переводил эти символы в свой язык единиц и нолей. Писать такие программы уже немного легче, хотя тоже пока очень тяжело: не очень понятно, надо следить за каждым регистром памяти, помнить в какой ячейке что лежит... Кроме того, у каждого процессора свой набор команд, а значит, и свой Ассемблер. Ассемблер считается языком низкого уровня, но, тем не менее, его изобретение сильно облегчило жизнь программистов. Теперь вместо ужасающих и длиннющих последовательностей нулей и единичек они могли писать просто команды из символов, отдалённо напоминающих обычный язык.

А вот языки высокого уровня уже созданы на, так сказать, сильно упрощённом человеческом языке (как правило, английском). Теперь компьютер сам решает, в какой регистр памяти загружать ваши данные, и как их обрабатывать, оставляя человеку более интересное занятие – быстро объяснить компьютеру, что он (человек, конечно) хочет от него (от компьютера, естественно), получить. И изучить эти языки значительно проще, чем, например, английский. Благо в любом языке программирования слов не очень-то и много – максимум пара сотен. И работают эти языки на большинстве современных компьютеров. Языков высокого уровня создано несколько сотен, и вот в конце XX века, появился, наконец, Питон.

Программы, созданные на Питоне, просты и даже, не побоюсь этого слова – элегантны, и, на удивление, кратки – то есть не надо набивать слишком много «букав» (как сегодня модно писать); а так как исследования показали, что программисты пишут примерно одинаковое количество строк кода каждый день независимо от языка, то программы на Питоне создаются несколько быстрее, чем на других языках программирования.

И вот вам пример:

Дан текстовый файл «24.txt» из 10^6 символов X, Y и Z. Надо определить длину самой длинной последовательности, состоящей из символов X.

Решение на Паскале (это тоже язык высокого уровня):

```
var maxLen, curLen, i: integer;
   s: string;
   f: text;
begin
 assign(f, '24.txt');
reset(f);
readln(f, s);
 maxLen := 1;
 curLen := 1;
for i:=2 to Length(s) do
if (s[i] = s[i-1]) and (s[i] = 'X') then
begin
    curLen := curLen + 1;
if curLen > maxLen then
     maxLen := curLen;
end
else
    curLen := 1;
writeln(maxLen);
end.
```

Решение на Питоне:

```
print(max(map(len, open('24.txt').readline().replace('Z', ' ').replace('Y', ' ').split())))
```

Ну как? Довольно лаконично, не так ли?

А поскольку Питон поддерживает **функциональный**, **императивный**, **объектно-ориентированный** и **процедурный** стили программирования... Как, вы не понимаете, о чём я? Да я и сам не понял, что написал... Вот после прочтения моего опуса залезете в Интернет и изучите все эти заумные стили, а пока – давайте научимся писать несложные (шутка! и довольно сложные тоже) программы на Питоне.

Язык программирования Питон существует примерно с 1991 года и с тех пор он вошёл в тройку самых популярных языков программирования. На Питоне написаны сотни тысяч программ, в том числе — Google, YouTube, Dropbox...

А теперь поговорим об *этом*... Поговорим о том, о чём в современном приличном обществе не говорят. Нет, не о том, о чём вы подумали! ...О зарплатах, конечно! Зарплаты у программистов на Питоне весьма велики, а больше я вам ничего не скажу – тема очень интимная.

Сейчас Питон – самый модный язык программирования, так что изучите его – и будете модниками. Но – «и на Солнце есть пятна»[1]...

Дело в том, что Питон – язык-интерпретатор, то есть каждая строчка программы превращается в машинные команды последовательно и тут же исполняется.

А вот, например, в компиляторах (Паскаль, C++) вся программа превращается в цепочку машинных команд и исполняется вся сразу. Отсюда недостаток Питона – программы на нём не могут выполняться так же быстро, как программы на полностью компилируемых языках, но программа («код», как сейчас модно говорить) пишется (и отлаживается) быстрее!

И тем не менее – нахождение всех простых чисел из массива чисел до 1 000 000 – а это 191 813 чисел! – занимает у меня на компе примерно 13 секунд!

Упражнение 0.1

Не удивлены? А как быстро вы досчитаете до 1 000 000? Ответьте навскидку!

Возведение 2 в 100 000-ную степень проходит за 0,2 секунды! А результат – это 30103 цифры! Невообразимо огромное число! Так что производительность у Питона вполне достаточная для бытовых целей ☺. Современные компьютеры считают так быстро, а новые задачи появляются так часто, что сейчас скорость разработки программы важнее её скорости выполнения, а программы на Питоне пишутся быстрее, чем на других популярных языках.

Кроме того, в Питоне существует огромное количество функций и методов, резко облегчающих и ускоряющих программирование, о которых я немного попытаюсь вам рассказать в последних главах для того, чтобы окончательно вдохновить/запугать вас.

И тем не менее, Питон – достаточно простой язык, конечно, не такой простой, как Фортран[2]... Извините, что-то меня на воспоминания потянуло...

1 Вы думали, что это сказал Галилей? А вот и нет! Выражение пришло из поэмы русского поэта, писателя и политического деятеляXVIIIвека М. М. Хераскова «Россияда»: «И в солнце, и в луне есть темные места!»

2 Фортран – один из самых первых языков программирования высокого уровня. Первый язык программирования автора

Но и не такой запутанный, как многие другие языки и вместе с тем достаточно мощный, чтобы быстро и эффективно решать достаточно сложные задачи. Ладно, хватить хвалить Питон, давайте на нём работать! Скачивать установщик Питона надо с официального сайта. Программа **бесплатная** (!).

Алгоритм следующий:

Заходим на https://python.org/downloads/windows/, скачиваем установщик (он называется Windows installer. В зависимости от установленной у вас на компьютере ОС скачиваем разные установщики! Что такое ОС, надеюсь, все знают?). Запускаем установщик с установками по умолчанию.

Всё, Питон готов к работе.

Да, а на чём мы будем писать программы? Кто сказал «на бумажке»? А, это вы, многоуважаемый дедушка Алгол Фортраныч… Времена, когда программы писались на бумажке, потом оператор забивал(а) их на перфокартах, потом перфокарты загружались в компьютер – прошли безвозвратно… Последний завод по выпуску перфокарт был закрыт в ещё прошлом веке; так что будем писать программы в специальной программе… (М-да, программа в программе под управлением программы Windows – эдакая матрёшка на современный лад). Программа (она же среда, она же редактор, она же оболочка!) называется IDLE (да, переводится как «Ленивый», но на самом деле названа в честь одного из создателей Питона). IDLE поставляется вместе с дистрибутивом Питона. Откройте стартовое меню Windows, выберите All Programs, в папке Python кликните по IDLE, и в результате видим что-то вроде вот такого (рис. 1):

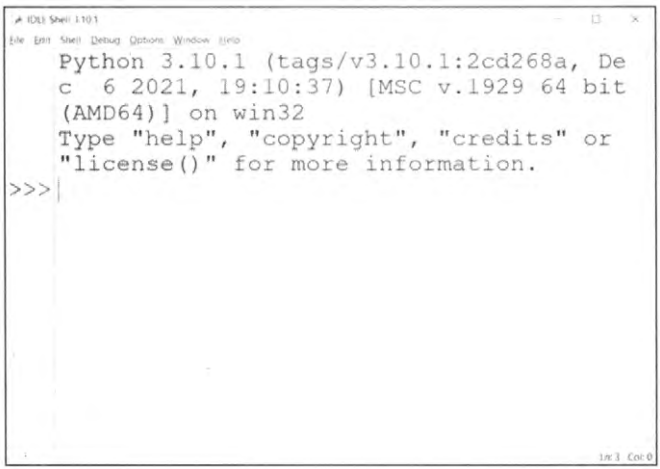


Рисунок 1

Это окно приглашает к **интерактивному** режиму работы, когда каждая введённая команда сразу выполняется и выводится на экран. В таком режиме можно писать и выполнять отдельные команды или выполнять очень простые программы.

Видите, после **>>>** мигает указатель |.

Это приглашение к работе. Проверим, как это работает.

Набираем:

```
>>>2+2
```

и жмём **Enter**.

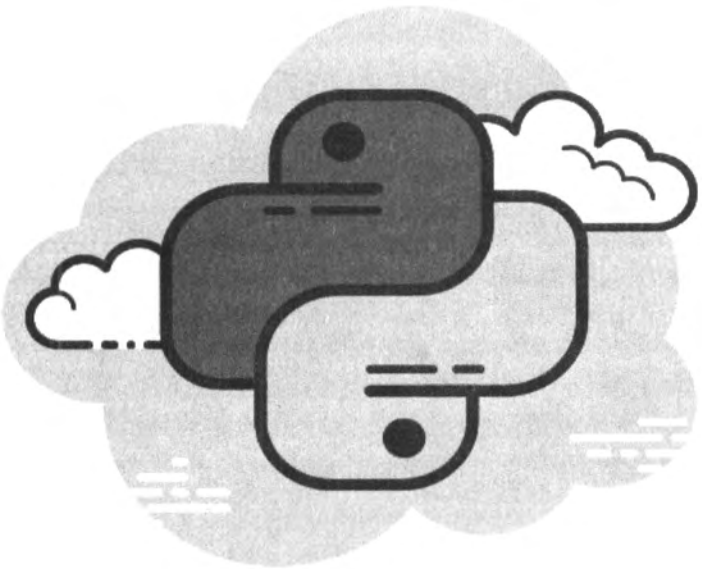
Видим:

```
4
>>>
```

Уррра! Работает! Ну всё, приступаем к работе. Как, я надеюсь, вы догадываетесь, главное действующее лицо в программировании – это **Переменные**.

Глава 1.

ПЕРЕМЕННЫЕ



Переменная *в программировании – это набор букв и цифр, значение которых может изменяться.*

Например, напишите в среде IDLE:

```
>>>X=1
```

Это означает, что переменной с именем **X** присвоено значение 1.

А теперь:

```
>>>X8=23
```

Другой переменной, с именем **X8** присвоено значение 23.

А теперь:

```
>>>X=3
```

Значение **X** стало равно 3. А теперь главная фишка программирования:

```
>>>X=X+1
```

Это не *уравнение*!!! Это значит, что к предыдущему значению **X**, равному 3, добавили 1 и теперь **X** равен 4. Убедимся в этом:

```
>>>X
```

(это приказ Питону показать значение **X**).

Ниже видим:

```
4
```

Это означает, что переменная **X** сейчас равна 4.

Выражения присваивания могут быть очень сложны, главное, чтобы слева стояла одна переменная, а справа может стоять сколь угодно затейливое выражение.

Например:

```
>>>X=(X*1.3127-X)/0.2
>>>X
6.254
```

Питон аккуратно и очень быстро считает: «сейчас **X** равен 4, значит, 4 умножаем на 1.3127, из результата вычитаем **X**, равный 4 и делим результат на 0.2, получаем 6.254, и поэтому теперь **X** равен 6,254» ! (символ * на программистском языке означает умножение).

Имена переменных в Питоне могут содержать только ***латинские буквы***, ***цифры*** и почему-то ***символ подчёркивания***.

Имя не может начинаться с цифры.

Питон чувствителен к регистру: **dig1, Dig1 и diG1** для него – разные переменные.

Русские буквы в имени переменной применять можно, но настоятельно не рекомендуется. Символ подчёркивания не следует ставить первым в имени.

Кроме того, в качестве имени переменной нельзя использовать вот эти 33 *ключевые слова*:

False	class	finally	is	return	none	continue
for	lambda	try	True	def	from	nonlocal
while	and	del	global	not	with	as
elif	if	or	yield	assert	else	import
pass	break	except	in	raise		

Ничего себе списочек! И как их всех запомнить? А и не надо... В любом случае, если есть сомнения, что это имя совпадёт с ключевым словом (а кроме того, нельзя использовать и *встроенные идентификаторы,* список которых я здесь не привожу, дабы сохранить психическое здоровье моих читателей), так вот, в таких случаях пишите имя с заглавной буквой, например **Class** или **eLSe** – и Питон спокойно и безошибочно воспримет это как переменную. Кроме того, эти ключевые слова, набранные в IDLE, загораются цветом, отличным от чёрного (это я немного вперёд забежал).

Упражнение 1.1

Какие переменные названы правильно:

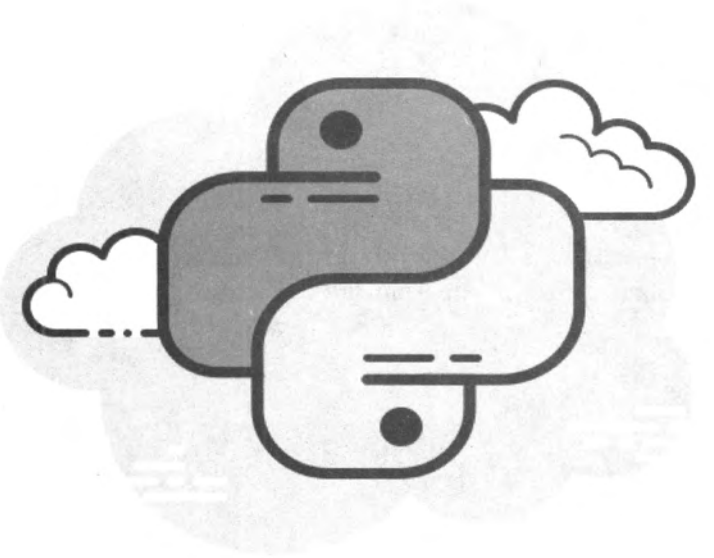
Out! , 3R, _3m, 3_m, Cat2y, DooШ, Almaz1996, TU-154, R____T, O600, import?

Мы познакомились с понятием ***переменной*** в Питоне. Имена переменных в Питоне могут содержать только ***латинские буквы***, ***цифры*** и ***символ подчёркивания***. Имя не может начинаться с цифры. Питон прекрасно различает большие и маленькие буквы.

Как-то совершенно незаметно и естественно в моём изложении промелькнули «**Арифметические операторы**».

Глава 2.

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ



Арифметические операторы в Питон такие же, как и в математике, вот только умножению не повезло. Это не крестик и не точка, а звёздочка.

Давайте посмотрим, как они работают.

Пусть **X** равен 3. Запускаем оболочку IDLE и в ней пишем:

```
>>>X=3
```

А переменная **Y** равна 8:

```
>>>Y=8
```

Теперь складываем **X** с **Y** и результат помещаем… э… ну, назад в переменную **X** (привыкайте, для программистов это совершенно обычное дело):

```
>>>X=X+Y
```

Ну и как узнать, чему равен **X**?!

Надо просто его напечатать:

```
>>>X
```

Ура, получилось – на экране:

```
11
>>>
```

Если вы желаете сохранить результат своих с Питоном действий – его (результат) надо обязательно присвоить какой-либо переменной. Давайте просто напишем:

```
>>>X+Y
```

На экране:

```
19
>>>
```

Не забываем, что сейчас **X** равен 11, а **Y** равен 8, так что результат правильный, вот только послушный Питон его никуда не поместил, приговаривая про себя: «Хозяин, я сложил **X** и **Y**... А куда поместить результат? Не указана переменная? Ну тогда помещу результат на экран, а дальше сам разбирайся...»

Проверим:

```
>>>X
11
>>>
```

Естественно, **X** не изменился!

Ну тогда займёмся вычитанием:

```
>>>Y=X-Y
```

И чему теперь равен **Y**?

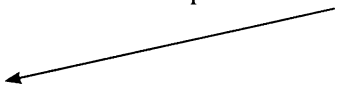
```
>>>Y
3
>>>
```

Всё правильно: 11-8 будет 3!

Умножение:

```
>>>Z=X*Y
```

Результат находится в новой переменной **Z**:



```
>>>Z
33
```

А теперь деление (сейчас **X** равен 11, а **Y** равен 3):

```
>>>Z=X/Y
>>>Z
3.6666666666666665
```

Да, переменные у нас получились двух типов: ***целые*** и …? Правильно, с ***плавающей запятой***.

Обратите внимание, что в качестве разделителя, по американской традиции, используется точка, а не запятая.

Поставите запятую – компьютер выдаст ошибку (в лучшем случае, в худшем – выдаст неправильный результат).

Обычные арифметические операции у нас закончились, так что переходим к необычным:

```
>>>Y=X//Y
```

Что это было – деление 2 раза подряд?! Да нет же – это просто целочисленное деление:


```
>>>Y
3
>>>
```

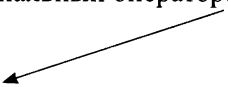
У нас **X** равен 11, а **Y** было равно 3, так что всё правильно. При выполнении этой операции просто-напросто отбрасывается вся дробная часть, как бы велика она не была, например, обычное деление:

```
>>>99999/10000
9.9999
```

А целочисленное деление даст:

```
>>>99999//10000
9
```

Ну и ещё один оригинальный оператор:



```
>>>33%10
3
```

Оператор **%** даёт нам ОСТАТОК от деления (**X** сейчас 11, а **Y** равен 3):

```
>>>Y=X%Y
>>>Y
2
>>>
```

Фууу… Ну вот теперь почти всё. Осталось только возведение в степень. Ну это просто:

```
>>>X**Y
121
>>>
```

X равен 11 , **Y** (не забыли?) равен 2, а 11 в степени 2 будет 121.

Приоритет выполнения арифметических операторов в Питон такой же как в математике – сначала умножение и деление, потом сложение и вычитание.

Поэтому результат следующего выражения…?

```
>>>A=2+2*2
```

Правильно, 6, а не 8! Мы получим 8, это если на калькуляторе посчитаем ☺.

Но операторов в Питоне в несколько раз больше, чем я пока перечислил, поэтому для ясности лучше пользоваться скобками (выражения в скобках выполняются в первую очередь).

Давайте закрепим и заодно проверим полученные знания:

Упражнение 2.1

Пусть x=2.5, а y=4.Сначала сосчитайте в уме, а потом в оболочке IDLE:

```
x=x*y
y=x-y
y=x//2
y=y%3
x=x/3
x=x//1
```

Упражнение 2.2

Ну а теперь задачка похитрее (пусть a=4, b=3):

```
a=(a+b)/2
a=a**2-2*b
b=b-a//(a-b)
a=a**0.5
```

Что за странный последний оператор? А, это просто квадратный корень из числа ***a***. Да, в Питоне можно смело возводить число в самые необыкновенные степени; в этом смысле Питон очень толерантный язык…

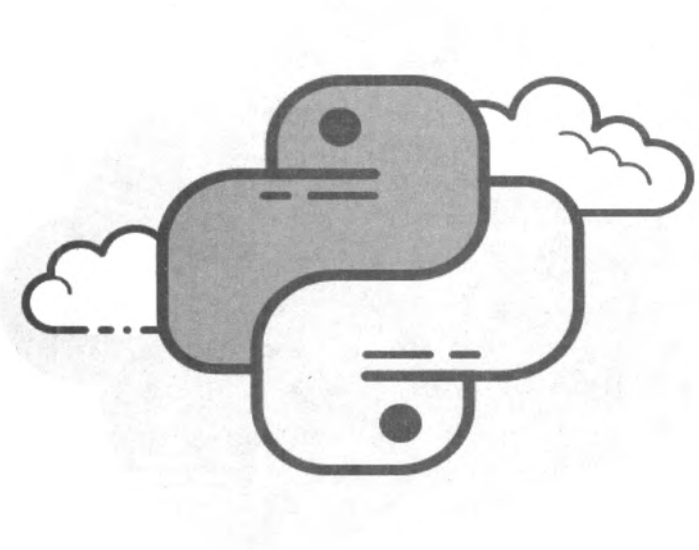
А арифметика в Питоне, такая же, как и в математике, вот только надо запомнить несколько новых символов: * – это умножение, ** – возведение в степень, // – деление нацело (без остатка), % – это как раз и есть остаток. Пока всё просто.

Я вижу, вам уже не терпится создать свою первую, настоящую программу. Хорошо! «И заметьте, не я это предложил![1]».

1 Цитата из бессмертного фильма «Покровские ворота»

Глава 3.

ПЕРВАЯ ПРОГРАММА



В оболочке IDLE (я надеюсь, она у вас запущена?) в меню **File** выберите пункт **New File** или, ещё проще, нажмите **Ctrl+N.**

Откроется новое окно (рис.2):

Рисунок 2

в котором мы и будем создавать программу. Какую? Кто сказал “Hello, world”? «Нет, мы пойдём другим путём[1]!»

Начнём с чего-нибудь простенького и математического (всё-таки компьютер в первую очередь нужен для сложных математических расчётов, а уже потом для игры в Мортал Комбат).

Напечатаем в открытом окне IDLE следующую программу (рис. 3):

1 Авторство этого бессмертного выражения приписывается В.И. Ленину

Рисунок 3

Для запуска программы нажимаем **F5.** Оп-па! Питон услужливо предлагает вам сохранить программу:

Source must be saved
OK to saved?

Что он сказал? Как не знаете английский?... Французский в школе учили? М-да, придётся вам немного подучить английский – Питон понимает только его. А Питон предупреждает: «***Источник (имеется в виду наша программулинка) должен быть сохранён. OK чтобы сохранить***».

Между прочим, это Питон мудро предлагает… Почему? Узнаете через несколько строчек. Нажимаем **OK**. Пока не сохраним, программа не запустится! Придумайте имя своей программе на латинице, например “Super”, “Great”, “Colossal”, и, главное, запомните папку, куда сохраняете. Сохраняем, и что мы видим на экране:

```
=RESTART:C:/Users/4820992/AppData/Local/Programs/Python/Python310/SUPER.py=
```

То есть Питон услужливо напомнил вам путь, где находится ваша программа и не более того. Э? А где результат? А где в нашей программульке приказ Питону: «Вывести результат на экран!». Кроме того, Питон не знает, ЧТО надо вывести на экран, для него все наши переменные равноценны. Хорошо, что сохранили программу, а то пришлось бы набивать всё заново. Мудр Питон, не по годам мудр…

Для вывода результатов на экран есть функция **print** («печатать» в переводе с английского).

Вернитесь в окно, где набирали программу, и добавьте в конце, со следующей строки:

```
print (q)
```

(именно так, с маленькой буквы! Ещё раз повторяю – Питон чувствителен к регистру!).

Рисунок 4

Нажимаем F5, сохраняем – и ура! – на экране, уже в другом окне – интерактивной работы (рис. 5):

Рисунок 5

видим результат – **111111111**.

Ну вот вам и первая, самая настоящая программа, которая считает произведение 12345679 x 9=111111111.

Ну это так, программа для детского сада (какие в детсадах сейчас дети умные – считают до ста одиннадцати миллионов сто одиннадцати тысяч ста одиннадцати!).

Поэтому теперь напишем программу для расчёта тормозного пути автомобиля **S** по несложной формуле:

$$S=\frac{v^2}{2g\mu}$$

Здесь:

- **v** – скорость машины в момент начала торможения, пусть это 60 км/час;
- **g** – ускорение свободного падения – 9,81 м/с^2;
- а вот **μ** – это коэффициент сцепления с дорогой, равный 0,7 (сухой асфальт).

Набираем программу (рис. 6):

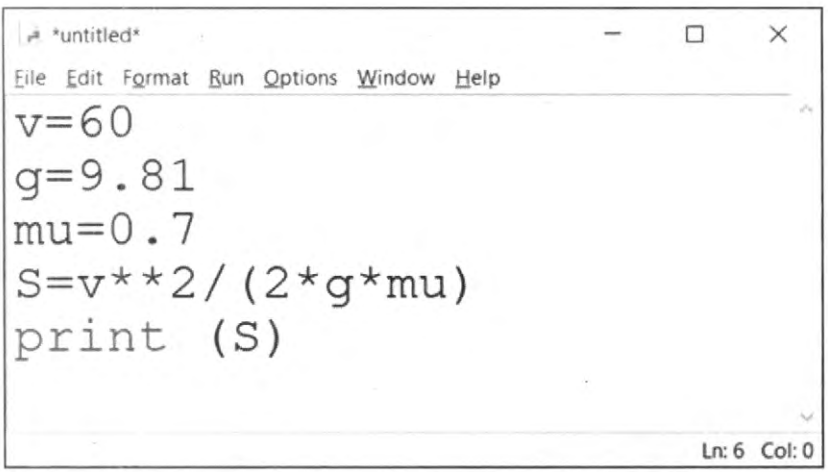


Рисунок 6

Обратите внимание, что знаменатель формулы тормозного пути я записал в скобках – чтобы было понятней, потому что без скобок эта формула выглядит несколько странно и запутанно: **S=v**2/2/g/mu.**Так что при малейшем сомнении ставьте безжалостно скобки – Питону всё равно, а нам, людям, понятней.

А где тормозной путь? Так вот же он:

```
262.12319790301444
```

262 метра?... Хм, многовато будет, для сухого-то асфальта… И вообще, а в чём здесь измеряется выведенная на экран величина? Непонятно… Дело в том, что мы скорость ввели в километрах в час, а ускорение – в метрах на секунду в квадрате (уже забыли школьные задачи по физике?), так что тормозной путь у нас получился, похоже, в метрочасах… Исправляемся: 60 км/час это 60 000 метров за 3600 секунд, или 16,6666666 м/с. Исправляем значение **v**, получаем вот такую программу (или «код», или «скрипт», выражаясь по-умному):

```
v=16.666666666
g=9.81
mu=0.7
S=v**2/(2*g*mu)
print (S)
```

Нажимаем-сохраняем-с удовольствием читаем:

```
20.22555539213307
```

Ну любит Питон точность! Тормозной путь 20 метров – вполне разумный ответ. Фуууух… наконец-то мы создали свою первую программу, причём:

при её создании мы прошли почти **все стадии** адской работы программиста – написание программы, исправление, проверка правильности работы, отладка и получение результата.

Ну что, кто после всех наших мук ещё желает стать программистом? Все?! Тогда вот вам для начала пренаипростейшее задание:

Упражнение 3.0

В 1935-1941 в Ленинграде работал Дом Занимательной Науки, в одной из комнат которого потолок был оклеен обоями в горошек, с числом горошин ровно 1 миллион. Естественно, большинство посетителей тут же начинали считать горошины с целью убедится, что их действительно миллион. Смешные люди! А мы с помощью Питона решим другую задачу: пусть эта комната была площадью, скажем, 125 квадратных метров. Оцените расстояние между горошинами.

А теперь немного посложней задача:

Упражнение 3.1

Пусть x=2, а y=3. Надо поменять значения местами, то есть сделать x=3, а y=2.

Это классическая простейшая программистская задача. А теперь задачи ещё сложнее:

Упражнение 3.2

Дано число T, показывающее, сколько минут прошло с начала суток. Надо вывести на экран показания электронных часов в этот момент в виде:

22 :18

Существенное осложнение: число T может быть больше, чем количество минут в сутках.

Упражнение 3.3

Напишите программульку, которая сообщит нам, до какой скорости (об/сек) надо раскрутить центрифугу с космонавтами, чтобы они испытали предельную перегрузку ~ 6g? (бОльшие перегрузки могут нанести непоправимые повреждения организму – мы же не хотим сделать из космонавтов инвалидов?).

Что? Как это посчитать? «Ищите и обрящете…[2]».

Ну? Нашли? Вот же она, формула – проще некуда:

$$A=\omega^2 \cdot R$$

Здесь:

- **A** – ускорение, равное 60 м/сек^2;
- **ω** – угловая скорость;
- **R**– радиус центрифуги (скажу вам по секрету, он составляет 7 метров).

Упражнение 3.4

Хороший марафонец пробегает дистанцию в 42 км 195 м[3] за 2 часа 10 минут. Просьба посчитать за сколько минут и секунд он пробегает в среднем каждый километр и сравнить со своим достижением на этой дистанции.

2 Евангелие от Матфея (глава 7). На самом деле цитат из Библии в нашем языке очень много, мы их просто не замечаем! Ну, например - «Не рой другому яму — сам в неё попадёшь!» - это переделанное выражение из Притчей Соломоновых , превратившееся в русскую пословицу

3 Согласно легенде, греческий воин в 490 году до н. э. после битвы при Марафоне, чтобы возвестить о победе греков, пробежал без остановки почти 40 км до Афин. Добежав, он успел крикнуть: «Радуйтесь, афиняне, мы победили!» и упал замертво. На первой Олимпиаде нового времени в Афинах решили провести забег от Марафона до Афин, и назвали это состязание – марафон. Длина забега сначала составляла примерно 40 км. В 1908 году на Олимпиаде в Лондоне королевская семья попросила продлить дистанцию, дабы старт проходил под балконом королевского замка. Длина удлинённой дистанции составила 42 км 195 м. С тех пор так и повелось…

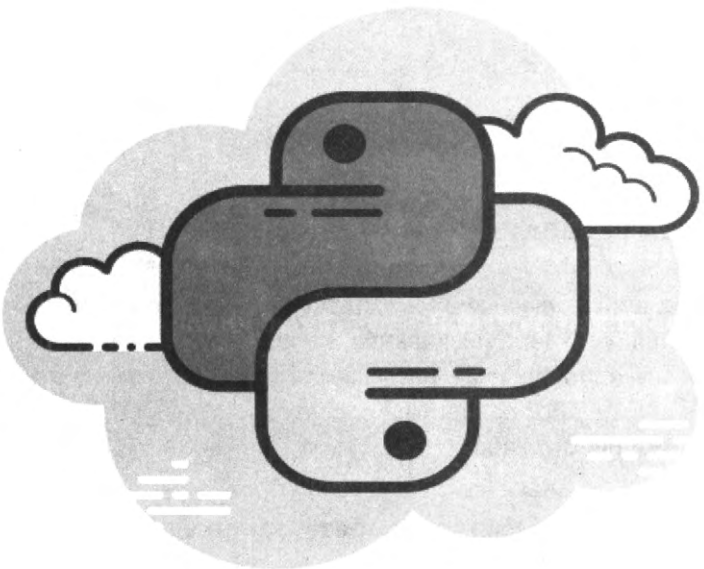
Ура! Написаны первые программы. Ничего сложного – нажимаем Ctrl+N, пишем программу, нажимаем **F5**, исправляем ошибки, любуемся на результат (если не забыли про оператор **print**). В оператор **print** можно вставлять текст и выполнять прямо внутри него вычисления (желательно не очень сложные, чтобы не усложнять восприятие программы).

Вы думаете, что в программировании только вычитают и умножают? Неправда ваша! На самом деле в Питоне есть ещё огромнейшее количество **операторов** для работы с числами, строками, множествами, списками.

Не переживайте – помаленьку узнаете о многом, но не обо всём!

Глава 4.

ЕЩЁ ОПЕРАТОРЫ



Питон – язык, "заточенный" под работу с текстами, поэтому сразу скажу про тип данных – "строка". Например, берём переменную **Y** и пишем прямо в IDLE:

```
>>>Y=121
```

Понятно, что Y сейчас равен 121:

```
>>>Y=-10.12
```

А теперь -10,12. А теперь так:

```
>>>Y='Грабаул! Караулят!'
```[1]

Как вы догадались, в кавычках записана строка (обыкновенный текст). Питон выполняет такое присваивание, не поморщившись! (кстати, кавычки могут быть как одинарные ' – называются **апостроф**, как в примере, так и двойные " – Питону всё равно; главное, чтобы строчка начиналась и кончалась кавычками одного типа).

Кавычки и апостроф набираются **обязательно** в английской **раскладке клавиатуры** – русские кавычки Питон не понимает (бормочет про себя при виде таких кавычек "Фу-фу, русским духом пахнет!").

1 А это цитата из рассказа В. Драгунского "Двадцать лет под кроватью"

Шучу, шучу – ничего он не бормочет. Пишет:

```
SyntaxError: invalid character '‘' (U+2019)
```

Что в переводе с питонного на русский и означает: "Фу-фу, русский кавичка! Не понимаем!". А апострофа в русской раскладке и вовсе нет…

В Питоне используется ***динамическая типизация***, то есть тип данных переменной изменяется в зависимости от присваиваемого ей значения.

И это показывает ещё один небольшой недостаток Питона – надо внимательно следить за типом переменных, иначе можно получить крайне неожиданный результат (в чём мы довольно скоро убедимся).

Вот вам примеры допустимых и недопустимых операций со строчными переменными:

```
>>>Y='Грабаул! Караулят!'
>>>D=2
>>>Ura='Ура?'
>>>Y=Y+Ura
>>>Y
'Грабаул! Караулят!Ура?'
```

Как видите, переменные, содержащие текст, можно успешно складывать.

И даже умножать:

```
>>> Y=Y*2
>>>Y
'Грабаул! Караулят! Ура? Грабаул! Караулят! Ура? '
```

Текст удвоился!

А вот делить и вычитать почему-то нельзя…

А вот если мы попытаемся сложить несложимое (D сейчас равно **числу** 2) :

```
>>>D=D+Y
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    D=D+Y
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

то Питон обижается! Говорит, что числа с буквами нельзя складывать! "Тщательней надо, ребята!" – как говаривал М. Жванецкий.

Упражнение 4.1

Надо напечатать на экране вот такую красотищу:

```
**********
*        *
*   +    *
*        *
**********
```

используя только строковые переменные, содержащие + и * и оператор умножения.

Кроме обычных, уже почти привычных нам ***операторов присваивания*** типа **d=d+5**, в Питоне есть и более непонятные операторы присваивания:

```
A+=1
```

Это означает A=A+1, всего лишь... Почему ещё нужен оператор **A+=1**?! Такая запись заставляет Питон создавать более компактный, быстровыполнимый код, чем обычная запись. Поэтому привыкайте; общая форма записи таких операторов (называется это **составной оператор присваивания**) выглядит так:

A☺=B⇔A = A☺B

где ☺ – любой(!) арифметический оператор.

Например:

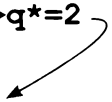
```
>>>q=10
>>>q+=2
```

теперь к **q** добавлено 2 и **q** стало равно 12.

```
>>>q-=4
```

а теперь **q** равно 8.

```
>>>q*=2
```



q равно 16.

```
>>>q//=3
```

А теперь **q** равно 5 (делили нацело!),

```
>>>q/=3
```

ну а теперь **q** равно 1.6666666666666667!

Точность Питона по умолчанию – 16 знаков после запятой!

С арифметикой разобрались. Теперь займёмся вводом – надо же как-то общаться с компьютером!

Для ввода данных с экрана имеется оператор **input** – в переводе означает "ввод".

Давайте его используем (результат ввода попадёт в переменную **Xx1**:

```
>>>Xx1=input()
```

Ну и что мы видим на экране? Ни-че-го! Компьютер ждёт ввода данных и ничего при этом не показывает! Это нехорошо! И вообще, зачем в **input** скобки?! А затем, что в скобки можно вставить любой текст (как на английском, так и на русском, а при должной сноровке – даже на китайском) и комп его послушно выведет!

Нажмите **2** и **Enter** – чтобы вернуть приглашение в IDLE (при этом переменная **Xx1** получила значение 2).

```
>>>
```

Печатаем в IDLE (текст, который желаете вывести на экран, надо заключать в двойные " или одинарные кавычки ' – причём эти кавычки печатаем только в *английской раскладке* клавиатуры):

```
>>>Xx1=input("Я жду число:")
```

Видим на экране:

```
Я жду число:
```

Печатаем **2** и нажимаем "ввод".

А теперь ещё:

```
>>>xX2=input("И ещё число:")
```

Вводим **3**. Сложим наши переменные:

```
>>>X=Xx1+xX2
```

И напечатаем их на экране посредством оператора **print** (да, в **print**, как и в **input**, можно вставлять любой текст):

```
>>>print ("сумма введённых чисел:", X)
```

Что мы увидим? Сюрприиииииз!

```
сумма введённых чисел: 23
```

Почему? Что за белиберда? 2+3=23?!

Как я и предупреждал выше, надо контролировать тип переменной… Дело в том, что **input** воспринимает введённое значение, как строку. Питон вполне обоснованно считает, что всё, что вводится с клавиатуры – это тексты! И тут он совершенно прав! Поэт же не обращается к нам с вот такими численными стихами:

```
14 126 14
132 17 43.
16 42… 511
704 83. [2]
```

Он пишет:

```
Не жалею, не зову, не плачу,
Все пройдёт, как с белых яблонь дым.
Увяданья золотом охваченный,
Я не буду больше молодым.[3]
```

Но мы отвлеклись. Питон воспринял введённые числа 2 и 3 именно как **текстовые** знаки! А нам надо число!

То есть надо введённое значение как-то превратить в число, причём целое. Для этого существуют функции преобразования типов.

Берём функцию **int** (от английского *integer*):

```
>>>X=int(X)
```

2 Прочтите не спеша, с грустным выражением лица, и вы поймёте, что это – пародия на С. Есенина. Взято из Интернета, по предварительным данным автор – П. Грайворонский

3 Ну а это уже точно С. Есенин

Помним, что **X** у нас 23 – как результат предыдущих вычислений. Узнаем, что получили:

```
>>>print(X)
23
```

Ой!... Ничего не изменилось… Это потому, что оператор **print** странным образом одинаково выводит и число, и строку…

Убедимся, что 23 – это число:

```
>>>X=X+X
>>>print(X)
46
```

Ура! Заработало!!!! А теперь проделаем всё заново, только в виде программы – введём 2 и 3 и сложим их. Вновь набираем в IDLE заклинание Ctrl+N и пишем программу. Операторы можно навешивать друг на друга (только не надо этим увлекаться – 2-3 оператора, не больше, иначе текст превращается в ребус):

```
Xx1=int (input("Я жду число:"))
xX2=int (input ("И  ещё число:"))
```

И ещё приятный момент: в **print** можно вставить арифметические (и не только арифметические!) операторы любой сложности:

```
print (Xx1+xX2)
```

Нажимаем **F5**, сохраняем программу под каким-нибудь именем на ваш вкус, вводим числа 2 и 3 и видим наконец-то искомый ответ:

```
5
```

А почему, собственно, вывод на экран выполняется с помощью оператора **print** ("печатать" в переводе)? О, это привет нам из 60-ых годов прошлого века… Дело в том, что раньше общались с компьютером посредством пер-

фокарт, а результаты выводились на бумагу электрической пишущей (то есть печатающей) машинкой…

Ещё раз повторю, что оператор **print** может выводить много переменных сразу, надо только переменные разделять запятой:

```
>>>print ('число X1=',Xx1,', число X2=', xX2)
```

И на экране читаем:

```
число X1= 2, число X2= 3
```

Стоп, а у нас ведь были переменные Xx1 и xX2? А что вы написали в тексте в операторе **print**, то компьютер и вывел! Компьютер и Питон выполняют только то, что им приказано и не умеют думать за вас (но это временно. Компьютеры умнеют буквально на глазах, и в этом есть вклад языка Питон).

Чаще, конечно же, надо вводить число не целое, а с плавающей запятой.

Ну это просто – по-английски плавающий – **float:**

```
>>>X1= float (input("Я жду число:"))
```

И введённое число 2 будет представлено как 2.0, а сумма **X1** и **X2** (а **X2**, если забыли, равно 3):

```
>>>print ('X1+X2=', X1+X2)
X1+X2=5.0
>>>
```

Кстати, обращаю ваше внимание: все операторы (я выделяю их **жирным** шрифтом) следует записывать так, как у меня и никак иначе! Никаких Print, flout и Input – всё это будет воспринято в лучшем случае как ошибки, а в худшем – как переменные с непредсказуемым результатом.

Давайте немного отдохнем и создадим простую программу:

Упражнение 4.2

Давайте поиграем в шпио... Нет, что вы! В разведчиков! Пусть у нас есть коробка, а в ней 15 рядов по 22 ячейки. И мы в неё фасуем по порядку обыкновенные фильтры для воды – 1-ый фильтр займёт 1 ряд, место 1, 23-ий – 2 ряд, место 1, 25-ый – 2 ряд, место 3. А в 174-ый фильтр мы спрятали сверхважное донесение, написанное невидимыми чернилами, задом наперёд да ещё и зашифрованное шифром Цеза... Ой, что это я вам все секреты выбалтываю? Про шифр Цезаря ещё будет задача!

Короче, надо сообщить в Центр, Алексу[4], в каком ряду и месте находится шифровка.

Наша самая первая программа получилась какой-то неказистой. Что-то там внутри себя проделала и выдала результат: 20.22555539213307! 20 чего? Килограмм? Парсек? Так сейчас программы не пишут! Программа должна разговаривать с клиентом и объяснять, что она получила.

Давайте для примера напишем красивую программку для вычисления площади круга по введённому диаметру. Примерно такую:

```
DiaKr = float(input ("Введите диаметр круга в см:"))
SqrKr=3.14*(DiaKr**2)/4
print (SqrKr)
```

Сначала оператором **input** вручную, с клавиатуры, вводим диаметр круга и помещаем значение в переменную, которую я назвал **DiaKr**. Желательно, чтобы название переменной намекало, что оно означает. Ладно, у нас первая программка из 3 строк. А если из 700? Допустим, назвали мы эту переменную D. А через 200 строк у вас обнаружились переменные D, d2, D7 и D_kr. Которая из них диаметр круга, вы за неделю работы над программой уже забыли... Так что лучше DiaKr.

4 Цитата из культового фильма "Семнадцать мгновений весны". Юстас - это секретный позывной советского разведчика Штирлица (он же полковник Исаев), а Алекс - это позывной советского руководства, которое давало Штирлицу новые задания

Теперь можно сосчитать площадь круга. Помните, чему она равна? Правильно, $\frac{\pi D^2}{4}$. Так как я (как будто) не уверен, что возведение в квадрат выполняется раньше умножения, я поставил для верности скобки. Нажимаем F5, вводим 1 и любуемся на результат:

```
0.785
```

Единственное замечание – всё-таки желательно указать, что за значение мы вывели:

```
DiaM = float(input ("Введите диаметр круга в см: "))
print ('Площадь круга диаметром',DiaM,'см составляет',
3.14 * (DiaM**2) /4,'квадратных сантиметров')
```

На экране:

```
Введите диаметр круга в см: 7.5
Площадь круга диаметром 7.5 см составляет 44.15625
квадратных сантиметров
```

Вот так-то оно понятней… Заметьте, в оператор **`print`** я вставил всё вычисление площади круга, тем самым сократив наш крошечный код ещё на 1 строку – программы на Питоне очень компактны.

Упражнение 4.3

Ну а теперь простенькое упражнение: надо написать программу (я бы даже сказал – программулечку), которая спрашивает "Как вас зовут", ждёт ваше имя и затем печатает "Привет, <ваше имя>!".

И сразу ещё пару заданий в догонку к предыдущему:

Упражнение 4.4

Усложним предыдущее упражнение. Пусть теперь программа запрашивает вашу фамилию, имя и год рождения, после чего выводит:

Здравствуйте, <имя фамилия>
Вам <x> лет

Где x – ваш возраст.

Упражнение 4.5

У Тургенева в рассказе "Му-му" указан рост Герасима – "мужчина 12 вершков роста, сложенный богатырём...". Открываем справочник: 1 вершок – 4,44 см, итого 12 вершков – 53,28 см... Богатырь?! 53 сантиметров роста?!

Дело в том, что в старые времена, говоря о росте человека, указывали его рост свыше 2 аршинов (1 аршин = 71,12 см).

Так что вот вам задача: программа запрашивает рост человека в вершках и выдаёт его рост в сантиметрах.

Иногда возникает необходимость записывать очень большие числа, ну, например, посчитаем, сколько *километров* отделяет нас от туманности Андромеды[5], находящейся, как известно, на расстоянии 2,52 миллионов световых лет от нас. А световой год – это расстояние, которое свет пролетает за 1 год. А скорость света – 300 000 километров в секунду. А в году 365 суток, в сутках 24 часа, в 1 часе 60 минут, в 1 минуте 60 секунд. У вас ещё не рябит в глазах от обилия цифр, которые надо перемножить? Тем не менее программа для расчётов получается довольно простой:

```
Vsveta=300_000
sutki=60*60*24*Vsveta
god=365*sutki
Tuman=2_520_000*god
print (Tuman)
```

5 Туманность Андромеды – самая известная Галактика, хорошо видимая невооружённым глазом в созвездии Андромеды, за что и получила своё название

Что за странная величина 300_000 присвоена переменной **Vsveta**?

При записи больших чисел очень хочется разделять разряды, а **пробел** в Питоне почти священный символ – им выделяются блоки; так что эту обязанность возложили на нижнее подчёркивание.

Например, 136 250 120 следует записывать или так:

136_250_120

или, если вы брезгуете использовать нижнее подчёркивание, как обычно:

136250120

Других вариантов нет!

Наберите программу у себя на компьютере, нажмите F5 !.. И на экране:

```
23841216000000000000
```

Вот это число! До галактики Андромеды 2,384 квинтиллиона километров… Воспринимать большие числа, записанные в таком виде, очень неудобно.

Но выход есть! Как известно, для записи очень больших или очень маленьких чисел используется специальная форма записи – "научная".

В этом случае число представляется в виде некоторой десятичной дроби, называемой ***мантиссой***, умноженной на целочисленную степень десяти (*порядок*).

В Питоне это записывается так: сначала пишется мантисса, затем пишется буква **e** (можно E, но обязательно в английской раскладке), затем пишется порядок. Пробелы внутри этой записи не ставятся. Так что расстояние до туманности Андромеды записывается как **`2.384122e19`**.

Причём если мы в нашей программе скорость света запишем в "научной" форме:

```
Vsveta=3e5
```

то и результат Питон выдаст в такой же, более удобной, форме:

```
2.3841216e+19
```

А если мне, наоборот, надо записать очень маленькое число, к примеру массу молекулы NH_3 (аммиак), равную $2{,}82 \cdot 10^{-26}$ кг?

Нет проблем:

```
2.82e-26
```

А если я хочу обычное число, ну например, удельный вес золота, записанный в обычной форме – 19320 кг/м^3 (представляете, литровая банка с золотом будет весить 19 кг!!! При том, что ведро воды весит 10 кг!) вывести в научной форме оператором **print**? Вот здесь создатели Питона дали маху, не придумав ничего лучше странного метода[6] **.format()** – методы в Питоне записываются вот так – через точку.

Его работу проще показать на примере:

```
>>>Au=19320
>>>AuOut='{:e}'.format(Au)
>>>AuOut
'1.932000e+04'
```

А почему число выведено в апострофах?

Вот в этом и состоит странность метода .format() – он преобразует число в строку!

6 Термин "метод" ещё не раз встретится в этой книге и в дальнейшей жизни. Метод – это способ, которым обрабатываются либо вырабатываются какие-либо данные

С другой стороны – ничего странного – все числа в Питоне содержатся в удобном ему (Питону) виде, а уж в какой форме вы хотите видеть число – объясните доступным Питону способом (извините, **методом**) и получите результат в удобной нам, людям, строке. Применять этот метод как-то не очень удобно, так как формат метода **.format()** в нашем случае довольно затейливый:

```
'{:.Ne}'.format(X)
```

где **N** задаёт количество знаков после запятой, апострофы намекают, что результат – это строка, фигурные скобки подсказывают Питону, где начало и конец формата, а **X** – преобразуемое число:

```
>>>Akk=0.000235698
>>>S='{:.2e}'.format(Akk)
>>>S
'2.36e-04'
```

Здесь я вывел число на печать с 2-мя знаками после запятой; по умолчанию выводится 6 знаков после запятой (ой, извините, точки):

```
>>>print ('{:e}'.format(Akk))
2.356980e-04
```

Удобнее метод **.format()** применять внутри функции **print()** – тогда выводимую на печать переменную можно использовать в дальнейшем; либо присваивать результат работы функции какой-нибудь другой переменной.

Помните **Упражнение 3.2** про электронные часы? Там у нас показания электронных часов печатались довольно странно:

```
5 : 0
```

Метод **.format()** позволит сделать вывод правильным, как и положено в электронных часах.

```
T=300
```

```
Min=T%60
T1=T//60
Hour=T1%24
print ('{:02d}'.format (Hour), ':','{:02d}'.format (Min))
```

и на экране мы видим, как и положено:

```
05 : 00
```

Обозначения `'{:02d}'` говорят Питону следующее:

- `2` – что надо вывести на экран не меньше 2 значащих цифр,
- `0` – недостающие разряды заполняем нулями,
- `d` – выводим число в целом виде.

Примеры работы этого формата:

```
>>>'{:02d}'.format (3365)
'3365'
>>>'{:07d}'.format (3365)
'0003365'
```

Число занимает 7 позиций, пустые места заполняем нулями:

```
>>>'{:17d}'.format (3365)
'             3365'
```

Число занимает 17 позиций, пустые места заполняются пробелами:

```
>>>'{:17f}'.format (3365)
'    3365.000000'
```

А вот здесь буква `f` означает формат с плавающей запятой, число занимает 17 позиций, по умолчанию выводится 6 знаков после запятой.

```
>>>'{:.2f}'.format (3365)
'3365.00'
```

Формат с плавающей запятой, 2 знака после запятой:

```
>>>'{:10.3f}'.format (3365)
'  3365.000'
```

Формат с плавающей запятой, 3 знака после запятой, число занимает 10 позиций.

Далее маленькое упражнение:

Упражнение 4.6

Требуется с помощью метода .format() вывести на экран число 2023 с 3 знаками после запятой, так, чтобы оно занимало 13 позиций, а недостающие места заполнить нулями.

На самом деле метод `.format()` имеет огромное количество разнообразных регулировок, которые я здесь приводить не стану, чтобы не разбудить моих уснувших читателей… Что? Ещё не заснули?

Тогда – кому любопытно, загляните в Интернет, а остальных приглашаю сделать упражнение:

Упражнение 4.7

Представьте, что Солнце мы уменьшили до размера футбольного мяча. Рассчитайте, до какого размера тогда уменьшится наша Земля и на каком расстоянии от Солнца она будет располагаться в таком масштабе. А где расположится Нептун – после разжалования Плутона он теперь самая далёкая от Солнца планета? И какой диаметр тогда будет у Юпитера?

Что? Каков диаметр футбольного мяча? Яндекс нам и вам в помощь… Сообщаю:

- **Диаметр футбольного мяча – 22 см.**
- **Диаметр Солнца – 1,39 миллиона километров.**
- **Диаметр Земли – 12742 километра[7].**
- **Диаметр Юпитера – 142984 километра (как учёные измерили Юпитер с такой точностью?!)**
- **Расстояние от Земли до Солнца 149,6 миллиона километров.**
- **Расстояние от Нептуна до Солнца – 4,55 миллиарда километров.**

Масштаб преобразования надо вывести в "научной" форме с точностью 1 знак после запятой, а остальные результаты (диаметры и расстояния) – в сантиметрах. Как вы думаете, уместится ли Солнечная система в таком масштабе в комнате? Квартире?

Какие интересные переменные в Питоне: **ди-на-ми-чес-кие**! Могут принимать какие угодно значения: и числа, и тексты… А ещё есть замечательный оператор **input**, чтобы компьютер мог объяснить, что он от меня хочет. А ещё переменные надо называть так, чтобы самому понятно было, что это за переменная. А ещё метод **.format** позволяет выводить число на печать в каком вам удобнее виде: можно в научном, можно с нулями впереди, можно сколько душе угодно цифр после запятой поставить.

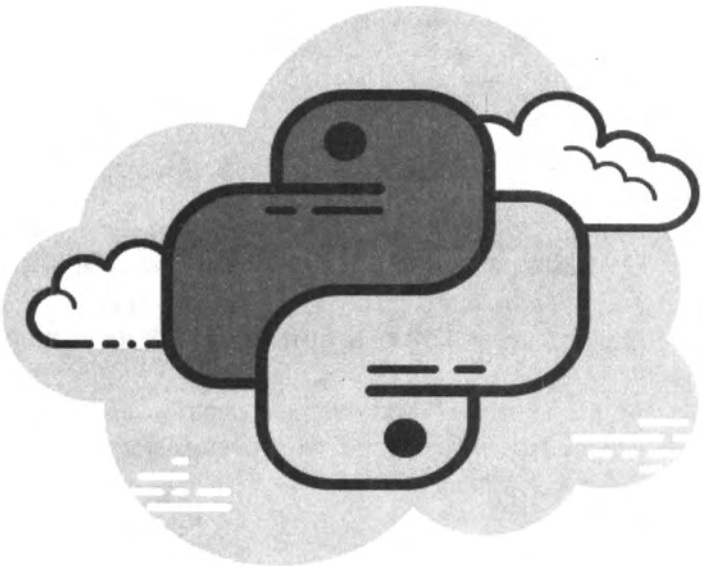
7 Кстати, если диаметр Земли уменьшить до 1,5 метров, то в таком масштабе толщина атмосферы (10 км) составит … 1 миллиметр! Шар, чуть ниже вашего роста и на нем тонюсенькая пленочка, в которой все человечество живет да еще и самолеты летают…

В нашей жизни мы очень часто вынуждены принимать решения в зависимости от обстоятельств: "сделать домашнюю работу или погонять в футбол?" или "если завтра понедельник, то начну новую жизнь".

Разумеется, процедуры принятия решений в зависимости от каких-то условий есть и в программировании.

Глава 5.

УСЛОВНЫЕ ОПЕРАТОРЫ



Ну вот, теперь можно перейти и к программированию. Что? А что это все было до этого? Воспоминания об арифметике и орфографии.

Программирование начинается с оператора **if**! (для не очень знающих английский: **if** переводится как "если").

Формат этого оператора такой:

```
if <логическое выражение> :
    <блок1: операторы, которые выполняем, если условие истинно>
else :
    <блок2: операторы, которые выполняем, если условие ложно>
```

Ну что? Это вам не 2 с 3 складывать! Данный оператор позволяет нам выполнить / или не выполнить какой-то участок программы неограниченной длины (назовём это **блок**) в зависимости от значения логического выражения.

Как вы, надеюсь, знаете, логическое выражение может быть только **True** (истина) или **False** (ложь).

Например, 2=2 это пример истинного выражения, а 2>3 – ложного. Вот только оператор = уже занят (это *оператор присваивания*). Поэтому оператор сравнения "равно" выглядит как ==

```
>>>print (2==2)
True
>>>print (2>3)
False
```

А вот оператор "не равно" выглядит изумительно (это как бы ≠, если кто не догадался):

```
>>>print (2!=3)
True
```

Приведу весь список простейших логических операторов:

- **<=** – меньше или равно
- **>=** – больше или равно
- **>** – больше
- **<** – меньше
- **==** – равно
- **!=** – не равно
- **in** – проверяет, входит ли элемент в последовательность (в переводе с английского означает "в")

Это простейшие операторы? А что, бывают и сложнейшие?

Да, бывают… И образуют они аж целый раздел математики – алгебру логики, или Булеву[1] алгебру. Но этим я вас попугаю немного позже.

***Блок** – это любое количество любых операторов, выделяемых одинаковым (обычно 4) числом **пробелов**.*

1 Джордж Буль — английский математик и логик. Один из основателей математической логики

Это важнейший элемент Питона! В Питоне именно пробелы (одинаковое число!) выделяют блоки, что позволяет, между прочим, сделать программу удобочитаемой и красивой. Конец блока – инструкция, перед который стоит меньшее число пробелов.

Ну а теперь составим первую программу с ветвлением (я даже не буду вам подсказывать: в среде IDLE нажмите Ctrl+N и набейте буквы, что написаны внизу):

```
A0=int (input(" введите число: "))
if A0 > 5:
     print(' больше')
else:
     print (' меньше')
```

Напоминаю, что **else** означает "иначе" – извините, всё время забываю, что, может быть, не все знают английский.

Введём число больше 5 – компьютер выведет "больше", меньше 5 – получим "меньше". Обратите внимание, что оба оператора **print** сдвинуты на равное(!) число пробелов. Это – важнейший момент в Питоне:

Сдвиг сообщает Питону о начале блока команд, идущих после двоеточия. Исчезает сдвиг – значит блок закончен.

А теперь усложним программу. Будем проверять, попало ли вводимое число между числами 5 и 10; так как Питон, в силу своей мягкости и терпимости к причудам программистов, разрешает указывать несколько условий сразу, то программу можно записать так:

```
A0=int (input('введите число: '))
if 5 <= A0 <= 10 :
    print('попали в заданный интервал от 5 до 10')
else:
    if A0 > 10:
        print ('больше')
    else:
        print ('меньше')
```

Как видите, выражение `5 <= A0 <= 10` Питон воспринимает спокойно (скажу вам по секрету – во многих других языках программирования такая конструкция немыслима).

Обратите внимание – выделение блоков пробелами улучшает читабельность программы.

Упражнение 5.1

Определите, что выдаст программа:

```
print(2==3)
print (0.001<0.001)
```

А теперь:

```
Logic=True
print (Logic)
```

А теперь похитрее:

```
L112a=(2>3)
print (L112a)
```

Все школьники мира ужасно страдают от квадратных уравнений. Ну, не от самих уравнений, а от решения этих самых уравнений. Сидит школьник на контрольной, решает уравнение… Но – "не решается задачка – хоть убей! Думай, думай голова поскорей!.."[2].

Упражнение 5.2

Школьники и взрослые! Давайте создадим программу для решения квадратного уравнения!

Программа получает на вход коэффициенты квадратного уравнения и выводит на экран найденные корни или "Уравнение не имеет корней!" в зависимости от значения дискриминанта.

2 Из чудесного стихотворение Марины Бородницкой

Да, надо вам сказать, что оператор **else** не всегда обязателен! Очень часто бывает, что надо что-то сделать, если условие выполнено, а если не выполнено – ничего не делать (ну, например, если сегодня понедельник, то начать новую жизнь, а если нет, то живём по-старому…).

Давайте создадим программу со странным названием "Не люблю нули". В соответствии с названием она вычёркивает в двузначном числе 0 и выводит полученный результат. Если нет нулей – число остаётся целым и невредимым. В Питоне программа получается очень простой:

```
z=int(input('Введите двузначное число: '))
if z%10==0:
     z=z//10
print (z)
```

Программа принимает введённое число, и проверяет остаток от деления этого числа на 10. У числа 99 этот остаток равен 9, поэтому ничего не делаем и выводим 99; а вот если ввести 90 – остаток от деления составит 0, мы поделим это число на 10 и выводим 9 (удалили 0!).

Упражнение 5.3

Напишите программу, запрашивающую у пользователя целое число и выводящую на экран информацию о том, является введённое число чётным или нечётным.

А теперь – задача похитрее. Напишем программу, в которую вы вводите диаметр… ну, скажем, подшипника. Если диаметр меньше 10 миллиметров, печатаем "БРАК!" (подшипник просто не налезет на ось), меньше 10,02 миллиметра – первый сорт, от 10,02 до 10,04 миллиметра – второй сорт, больше 10,04 – опять-таки "Брак!" – подшипник болтается на оси… Программа выглядит примерно так:

```
x=float(input("Введите диаметр детали: "))
if x<10.0 :
    print ("БРАК!")
```

```
else:
    if x<=10.02:
        print ("1 сорт")
    else:
        if x<=10.04:
            print ("2 сорт")
        else:
            print ("БРАК!")
```

Как видите, очень часто после **else** идёт следующий **if**. Это и в самом деле очень частое явление в программах, и создатели Питона на это отреагировали, введя оператор **elif**, объединивший конструкции **else : if** в одну. Переделаем программу, используя **elif**:

```
x=float(input("Введите диаметр детали: "))
if x<10.0:
    print ("БРАК!")
elif x<=10.02:
    print ("1 сорт")
elif x<=10.04:
    print ("2 сорт")
else:
    print ("БРАК!")
```

Программа стала короче, а, главное – удобочитаемой.

Упражнение 5.4

Существует множество достаточно занудных задач, где проделывается множество однотипных и скучных действий (пример – бухгалтерские задачи). Вот давайте и решим довольно занудную задачу по расчёту дня недели 2023 года по количеству дней, прошедших с начала года. Начальные условия: 1 января 2023 пришлось на воскресенье. Конечные условия: получив от пользователя число дней, прошедших с начала года, печатаем расчётный день недели (вот здесь оператор *elif* очень даже пригодится).

Упражнение 5.5

Напишите программу, которая запрашивает у пользователя, сколько у него денег (в рублях) и выводит на экран надпись, например: "У вас 5 рублей" в правильной форме. Мы ведь говорим 1 рубль, 2 рубля, 3 рубля, 4 рубля, но 5 рублей, 6 рублей. А почему?

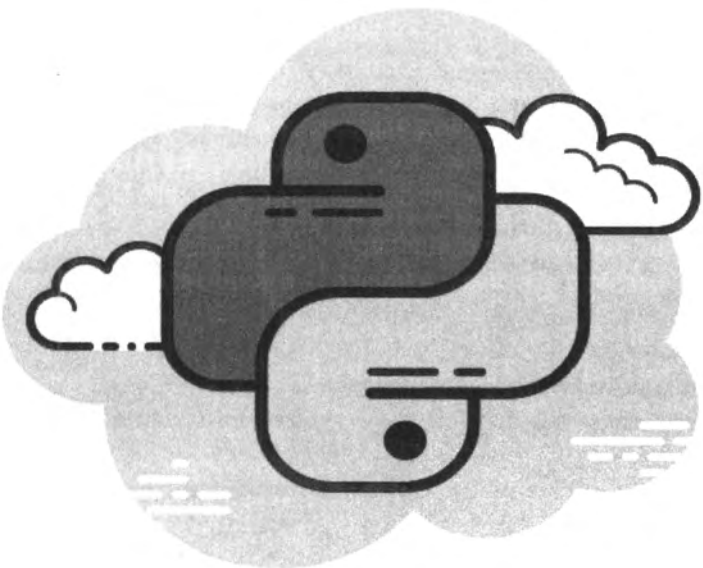
А потому, что в старославянском языке существовала форма двойственного числа, которая таинственным образом распространилась и на числительные три и четыре. Так что, получив на вход целое число, программа должна вернуть его вместе с правильной формой слова рубль.

Оператор **if/else** позволяет в зависимости от заданных условий выполнять, или, наоборот, обходить определённые участки программы. А ещё есть оператор **elif** – с его помощью можно проверять дополнительные варианты основного условия. И самое главное – очень часто ведь бывает, что если условие не выполнено, то можно ничего не делать. Если рак на горе не свистнул – сидим на… э…стуле ровно и оператор **else** не используем.

Пока в операторах **`if / else`** мы применяли простенькие логические выражения. Но ведь я обещал, что бывают и более сложные логические выражения!

Глава 6.

ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ



Вот, допустим, вы помощник депутата по формализации обещаний – ну, чтобы депутат не забывал исполнять свои обещания (ну автор и фантаст! Это где ж такое бывает?). Отвечаю: "В некотором царстве, в некотором государстве, жил да был депутат, иногда забывающий выполнять свои обещания...". Записываем алгоритм:

```
if (вчера был четверг):
    if (вчера был дождь):
        print ('приступаю к выполнению обещаний')
```

Как видите, для выполнения обещаний надо, чтобы одновременно 2 события были истинны: и четверг, и дождь. В Питоне союз "и" носит гордое имя "логический оператор" и пишется как **and** (и совсем уж по-умному именуется "конъюнкция" и на языке математики обозначается ∧).

Оставим депутата в покое, а для примера напишем программу, которая открывает перед Джеймсом Бондом (который агент 007)[1] все двери:

```
Name=input('Имя:')
Surname=input('Фамилия:')
Number = input('№ удостоверения агента:')
                                                    (6.1)
if Name=='Джеймс' and Surname=='Бонд' and Number== '007':
    print ('Все двери открыты')
else:
    print ('Ходят тут всякие...')
```

1 Джеймс Бонд - главный персонаж романов британского писателя Яна Флеминга и фильмов, снятых по ним, — агент МИ-6, также известный как "агент 007"

Так, я чуть не забыл, что Джеймс Бонд англичанин и по ошибке может набрать своё имя-фамилию по-английски.

Придётся модернизировать программу:

```
TXT='Ходят тут всякие...'
Name=input('Имя:')
if Name=='Джеймс' or Name=='James':
    Surname=input('Фамилия:')
                                                    (6.2)
    if Surname=='Бонд' or Surname=='Bond':
        Number = input('№ удостоверения агента:')
        if Number=='007':
            TXT='Все двери открыты'
print (TXT)
```

Здесь применяется другой логический оператор – `or` (в переводе с английского – "или" и гордо именуется "дизъюнкция"; на языке математике обозначается ∨).

А теперь двери открываются, если ввести "Джеймс" или "James" и "Бонд" или "Bond".

Логические операторы можно навешивать друг на друга (тут главное – не запутаться!):

```
Name=input('Имя:')
Surname=input('Фамилия:')
Number = input('№ удостоверения агента:')
                                                    (6.3)
if Number=='007' and (Name=='Джеймс' or Name=='James') and
(Surname=='Бонд' or Surname=='Bond'):
    print ('Все двери открыты')
else:
    print ('Ходят тут Бонды всякие...')
```

Мне больше нравится вариант программы 6.2 как более понятный и закрывающий двери перед всеми подделками под Джеймса Бонда при малейшей ошибке во вводе.

Упражнение 6.1

А теперь напишите программу, которая пропускает в сказочный мир только Илью Муромца, Добрыню Никитича или Алёшу Поповича. Сначала спрашиваем фамилию и кличку, а потом решаем – пускать или нет.

Давайте сделаем таблицу значений операторов **or** и **and**.

Логические значения **True** и **False** часто для упрощения заменяют на 1 и 0 соответственно.

Так что напишем вот такую программу:

```
T=1
F=0
print ('0 or 0 =',F or F)
print ('0 or 1 =',F or T)
print ('1 or 0 =',T or F)
print ('1 or 1 =',T or T)

print ('0 and 0 =',F and F)
print ('0 and 1 =',F and T)
print ('1 and 0 =',T and F)
print ('1 and 1 =',T and T)

print ('not 1 =',not T)
print ('not 0 =',not F)
```

А что за странные последние 2 строчки?

А это ещё один логический оператор **not** (в переводе – "нет", и в математике обозначается несколько странно: ¬), который просто инвертирует (переворачивает) логическое значение, делая из **True – False**, а из **False - True** (прямо как некоторые политики!).

На экране:

```
0 or 0 = 0
0 or 1 = 1
1 or 0 = 1
1 or 1 = 1
0 and 0 = 0
0 and 1 = 0
1 and 0 = 0
1 and 1 = 1
not 1 = False
not 0 = True
```

(Питон иногда путает 1 с **True,** а 0 с **False**...) Эту таблицу следует распечатать, повесить на кухне и выучить наизусть – эти данные нам ещё пригодятся.

Упражнение 6.2

Что мы увидим на экране после выполнения этой программы:

```
print (2>3 and 1<=4)
print (2>3 or 1<=4)
```

А теперь:

```
a=2<3
print (not a)
b=a and True
print (b)
c=2>3
e=a or b and c
print (e)
```

А теперь давайте сами напишем несложную программу, распутывающую хитрые логические комбинации.

Упражнение 6.3

Составьте таблицу истинности для очень-очень запутанной логической операции, записанной математическим языком:

$X \wedge \neg Y \vee \neg X \wedge Y$

Таким образом, здесь главное правильно записать это логическое выражение в обозначениях Питона, а остальное программа сделает сама.

Помните, мы с вами проверяли, попадает ли введённое с клавиатуры число в промежуток [5;10]?

Запутаем задачу.

Упражнение 6.4

Вводим с клавиатуры 2 числа. Каждое из этих чисел должно быть либо меньше 5, либо больше 10. Если условие выполняется, печатаем "числа подходят", не выполняется – "числа не подходят".

Вопрос из зала: "А если подходит только одно число?". Отличный вопрос! Что ж, мы "хороших перспектив никогда не супротив[2]" – печатаем "подходит только 1 число".

Между прочим, Питон позволяет выполнять над логическими значениями и более хитрые операции! Он умеет сравнивать логические выражения, исходя из постулата "правда больше, чем ложь".

Запишем программку:

2 Строчка из великолепной и смешной поэмы Л. Филатова "Про Федота-стрельца, удалого молодца"

```
T=True
F=False
print ('False <= False =',F <= F)
```

Да, здесь оператор **<=**, как ни странно, самое обычное сравнение "меньше или равно":

```
print ('False <= True =',F <= T)
print ('True <= False =',T <= F)
print ('True <= True =',T <= T)
```

Читаем на экране:

```
False<= False = True
False <= True = True
True <= False = False
True<= True = True
```

Это мы составили таблицу истинности хитрой логической операции под названием "Импликация" (математическое обозначение – → (стрелочка вправо), а в Питоне обозначается как бы стрелочкой влево <=)!

С помощью Питона очень удобно решать довольно затейливые логические уравнения, например, вот такие простенькие:

Упражнение 6.5

Для какого целого значения X истинно высказывание:

¬ ((X>2) → (X>3))?

Задачка проще некуда? Можно и без программы решить?

Хорошо, вот задача посложнее:

Упражнение 6.6

Найти все целочисленные решения уравнения

((X<25) → (X<23)) ∧ ((X<22) →(X>21))

из отрезка [20,24].

А теперь ещё похитрее:

Упражнение 6.7

Для каких целых чисел из интервала (40;60) истинно логическое условие:

¬((X кратно 5) → (X кратно 25))?

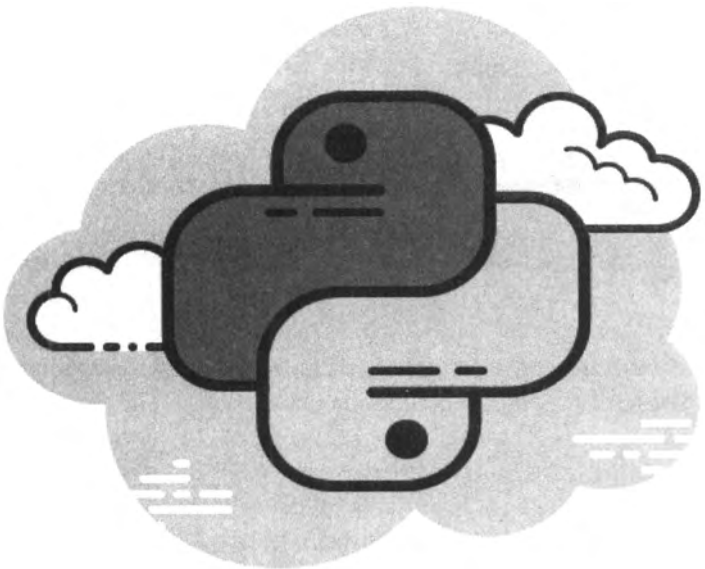
Интервал (40;60), между прочим, удобнее записывать так:]40;60[.

Очень полезны операторы **and** и **or**, позволяющее проверять одновременность выполнения/невыполнения логических условий. Ну и конечно, есть ещё оператор отрицания **not**, который всё отрицает – прямо как некоторые граждане возраста N, где (N < 7 or N > 70).

Ну что, все операции изучены? Нет, что вы! Остались, можно сказать, одни из самых главных понятий в программировании – **циклы**.

Глава 7.

ЦИКЛЫ



Очень часто какую-то часть программы надо выполнить неоднократно.

Например, у нас есть список из 107 112 человек и нам надо найти в нём какого-нибудь Айвазовского. Тогда нам надо написать программку для поиска определённой фамилии и запустить её всего-то 107 112 раз, сверяя каждую взятую из списка фамилию с фамилией "Айвазовский". Такая операция и называется ЦИКЛОМ. А выглядит цикл так (**`for`** – "для", **`in`** – "в"):

`for` <переменная > **`in`** <последовательность >:
<тело цикла>

Ну, с переменной всё ясно – это обычная переменная, которая по одному принимает значения из *последовательности*.

Последовательность *– это набор букв и/или чисел с определённым порядком следования.*

То есть (1,2,3) – это одна последовательность, а (2,3,1) – другая.

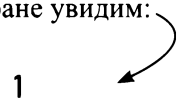
Тело цикла *– это любое количество операторов, которые выполняются при каждом шаге цикла и выделяются равным числом пробелов (рекомендуется 4 пробела).*

Кстати, IDLE автоматически вставит 4 пробела после того, как вы наберёте двоеточие – главнейший, хотя и незаметный элемент цикла!

Запишите и запустите простенькую программку (помним про двоеточие! А пробелы после двоеточия IDLE вставит сам):

```
for dgt10 in (1,3,7.2,'Fa',13):
    print (dgt10)
```

На экране увидим:



```
1
3
7.2
Fa
13
```

Как видите, последовательность не обязательно должна состоять из чисел.

Тем не менее, в большинстве случаев переменная цикла принимает числовые значения с равным шагом, то есть пробегает последовательность типа такой (0, 1, 2, … , 99, 100).

Как записать такую последовательность? Вот так прямо, в скобках перечислить все числа? А если их миллион? 2 месяца записывать будем? Вот уж нет – лень – двигатель прогресса![1]

Для этой цели есть простенькая функция **`range`** ("ряд" по-английски). С её помощью последовательность (0, 1, 2, … , 99, 100) записывается очень просто – **`range(101)`**! Перебор чисел в функции **`range`** заканчивается перед числом, указанным в скобках.

1 Вот уж никогда бы не подумал, что автор этой сентенции – поэт Андрей Вознесенский! "Лень – двигатель прогресса" – это строчка из его стихотворения "Лень"

А если такая же последовательность начинается с 1?

Пожалуйста – **`range(1, 101)`**.

А для последовательности (3, 5, 7, … , 30, 33)?

Например, вот так: **`range(3, 34, 2)`**. Значит, формат функции **`range`** такой:

```
range ([ начало, ] конец [ ,шаг ])
```

В квадратных скобках указаны необязательные параметры.

> Обязательно указывать только параметр ***конец***, в этом случае параметр ***начало*** равен 0.

Почему 0, а не 1, как у всех нормальных людей?

Дело в том, что на Питоне работают в основном математики и программисты, а это не совсем нормальные люди. У них счёт начинается с нуля – вспомните вывод в конце упражнения 4.2.

> И нумерация значений в последовательностях в Питоне начинается с 0, и функция **`range`** "заточена" именно под это, и, чтобы пробежать все элементы в последовательности (0, 1, 2, 3, 4) надо записать **`range(5)`**.

Опять-таки, почему не **`range(4)`**? Потому, что значение *конец* не включается в создаваемую последовательность, что опять-таки связано с нумерацией последовательностей – так удобнее.

Например, если в последовательности 5 элементов, то, чтобы перебрать все – пишем просто **`range(5)`**:

```
for i in range(5):
    print (2*i)
```

А на экране видим фрагмент таблицы умножения на 2 – первые 5 строчек:

```
0
2
4
6
8
```

Вот ещё пример:

```
for T_T in range (-3,2):
     print (T_T, end=', ')
```

На экране мы увидим:

```
-3, -2, -1, 0, 1,
```

Тут всё просто, кроме странного параметра **end**=. Что за зверь такой?

Параметр **end=** задаёт символ конца строки (по умолчанию это символ "\n", который в Windows автоматически превращается в "\r\n" (переход в начало следующей строки)).

И не спрашивайте меня, почему для этой простой операции используются 2 символа – для этого мне надо начать объяснение с прошлого века… Нет, не с XIX века, что вы!.. С конца двадцатого века. В эти незапамятные времена вывод данных, как вы помните, происходил с помощью электрических пишущих машинок, а в них для перехода в начало следующей строки выполнялись 2 команды: сначала каретка с бумагой просто сдвигалась вправо до упора, а затем бумага сдвигалась на строку ниже.

В данном примере удобнее вывести результаты одной строкой, поэтому я попросил Питон в конце вывода ставить запятую (и пробел).

А теперь добавим в эту программку ещё строку:

```
for t_T in range (-3,2):
     print (t_T, end=', ')
print ('*' * 50)
```

Добавленным оператором я хочу подчеркнуть вывод 50-ю звёздочками. На экране:


```
-3, -2, -1, 0, 1, **************************************************
```

Не совсем то, что я ожидал! В таких случаях следует вставлять "пустой" оператор **print()**, который переведёт указатель начала печати в начало следующей строки:

```
for t_T in range (-3,2):
     print (t_T, end=', ')
     print()
print ('*'*50)
```

Да что ты будешь делать! Теперь на экране:

```
-3,
-2,
-1,
0,
1,
**************************************************
```

Вот вам пример того, как Питон определяет конец цикла – у нас **print()** попал в тело цикла, так как перед ним стоит 4 пробела, и Питон послушно перескочил на новую строчку 5 раз.

Удалим пробелы перед **print()**:

```
for t_T in range (-3,4):
     print (t_T, end=', ')
print()
print ('*' * 50)
```

Ну наконец-то:

```
-3, -2, -1, 0, 1,
************************************************
```

И ещё пример:

```
for i in range (40,10,-5):
     print (i, end='+')
```

Да, шаг может быть и отрицательным!

На экране мы увидим:

```
40+35+30+25+20+15+
```

Упражнение 7.1

На основе предыдущего задания напишите программку, которая выводит на экран запись:

13+20+27+34+41+48+55 =

а после знака равенства поставьте, чему равна эта сумма...

Последовательности в Питоне могут иметь совершенно изумительный вид, типа такого:

```
Word='Катя'
for j in Word:
    print (j*2, end='')
```

Результат:

```
ККааттяя
```

То есть Питон переменную **Word**, содержащую слово **Катя,** воспринял тоже как последовательность (!), совершенно спокойно умножил каждую букву на 2 (то есть просто удвоил) и вывел результат на печать. Напишите **print(j*3)** – произойдёт утроение букв.

Давайте немного передохнём, выполнив простые упражнения:

Упражнение 7.2

Даны 2 последовательности (0, 4, 7, 2, 1, 19, 6) и (18, 1, 9, 3, 4). Надо найти совпадающие элементы и вывести их на экран, разделив пробелом.

Упражнение 7.3

Напишите программу, которая из заданного массива (-2, 5, 8,-4, -6, -11) выводит модули значений элементов.

Перейдём к более хитрому виду цикла – ***циклу с условием***, или циклу **while (while** по-английски "пока" (не в смысле "до свидания", а в смысле "до тех пор, пока")).

Определяется этот цикл так:

```
while < логическое выражение >:
      < тело цикла >
```

Тело цикла выполняется до тех пор, пока *логическое выражение* истинно, то есть равно логическому **True**. Проще показать на примере:

```
N=1
while N<=10:
      print(N, end="*")
      N=N+1
```

Результат очевиден: 1*2*3*4*5*6*7*8*9*10*. Как только **N** достиг 11, программа прервалась и тело цикла не выполнилось – тут всё просто.

А вот что случится, если мы забудем про пробелы в **N=N+1**?

```
N=1
while N<=10:
     print (N, end="*")
N=N+1
```

В результате Питон будет выводить бесконечную последовательность единиц – ведь N всегда будет равен 1, и выхода из цикла не случится! Вот это и есть основная трудность цикла **while** – надо следить, чтобы условие выхода из цикла выполнялось хотя бы иногда! Чтобы остановить Питон, нажимаем Ctrl+C (в английской раскладке!).

Ну вот, теперь пойдут задачи посложнее.

Упражнение 7.4

Что напечатает программа?

```
s = 0
n = 1
while s < 100:
    s = s + 20
    n = n * 3
print(n)
```

Упражнение 7.5

Создайте программу, которая запрашивает у пользователя пароль. Если пароль ошибочный, то печатает "неверный пароль" и вновь запрашивает его. Если пароль верный – печатает "Вход разрешён" и завершает программу. Что? Какой пароль? Ну, допустим, 123.

Иногда бывает нужно внутри цикла пропустить какую-либо операцию, либо резко прервать цикл. Немного непонятно? Приведу пример:

```
N=0
while True:
    N+=2
    if N==6:
        continue
    if N==20:
        break
    print (N, end=' ')
print ('\n I'll be back')
```

Разберём программу. Сначала переменной **N** присвоим значение 0. Затем запускаем бесконечный (!) цикл. Да, когда мы не знаем, в какой момент выполнения цикла случится ожидаемое событие (но вы уверены – это всенепременно случится), можно запускать цикл **while** со всегда истинным условием. Дальше на каждом обороте цикла увеличиваем **N** на 2 и выводим результат на печать. Но! Но когда **N** станет равным 6, стоит оператор **continue** (в переводе означает "продолжать"). Питон послушно прыгает вновь в заголовок цикла, проверяет всегда истинное условие, вновь увеличивает **N** на 2 – а число 6 при этом не было выведено на печать – и мы видим на экране:

```
2 4 8 10 12 14 16 18
I'll be back
```

Когда **N** достигает 20, Питон выполняет команду **break** (вы правильно догадались – в боксе есть такая же команда – брейк – и означает она "прекратить").

Прекратить так прекратить – Питон прекращает выполнение цикла и выводит *I'll be back.* (Все смотрели фильм "Терминатор"? Все?! Странно, я-то не смотрел, но почему-то знаю эту фразу…).

Маленькие нюансы к последнему оператору – что за \n стоит в начале бессмертной фразы Шварценеггера?

А это **символ перевода строки**, я о нём писал выше. Я хочу напечатать "I'll be back" на новой строке, а невидимый указатель точки вывода стоит после числа 18, помните, что означает **end=' '** в операторе **`print`**?

А второй нюанс – как вывести на экран **апостроф**?

А вот для этого в Питоне для вывода строки и предусмотрены символы " (кавычки) и ' (апостроф). Всё очень просто – надо напечатать апостроф – берём строку в кавычки; надо кавычки – берём строку в апострофы…

Упражнение 7.6

Переместите в приведённой выше программе 1 строку так, чтобы последним выведенным числом было 20, а бессмертную фразу I'll be back **напечатайте на любом другом иностранном языке, желательно на китайском или японском.**

В детстве мне очень нравилось решать математические ребусы, ну, то есть задачи типа таких:

```
  ВАГОН
+ ВАГОН
 СОСТАВ
```

Здесь разным буквам отвечают разные цифры, надо определить какая цифра соответствует какой букве.

Конечно, эти задачи раньше решались путём сложных логических рассуждений, а вот мы, вооружённые знаниями по программированию, решим такую задачу "в лоб". Допустим, дана задача:

```
  ОДИН
+ ОДИН
 МНОГО
```

Здесь всего 6 разных букв, поэтому нам придётся запустить 6 циклов, перебирающих для всех букв цифры от 0 до 9. Самые сообразительные уже машут руками, крича: "Нет! По букве М можно цикл от 1 до 9! Нет, даже от 1 до 1, ой, то есть просто 1!" А потом надо сделать множество проверок, чтобы все буквы были разными цифрами. А потом, уже в конце, надо из полученных цифр собрать числа, например:

ОДИН=О*1000 + Д*100 + И*10 + Н

и проверить равенство: ОДИН + ОДИН = МНОГО. Получилась у меня вот такая программа:

```
m=1
for o in range(10):
    for d in range(10):
        for i in range(10):
            for n in range(10):
                    for g in range(10):
                            if o==d or o==i or o==n or o==m or o==g:
                                continue
                            if d==i or d==n or d==m or d==g:
                                continue
                            if i==n or i==m or i==g:
                                continue
                            if n==m or n==g:
                                continue
                            if  m==g:
                                continue
                            odin=o*1000+d*100+i*10+n
                            mnogo=m*10000+n*1000+o*100+g*10+o
                            if (odin+odin) == mnogo:
                                print (odin,'+',odin,'=',mnogo)
```

(Обратите внимание, как красиво и интересно программа смотрится издалека – как картинка).

Запускаем и мгновенно имеем результат:

```
6823 + 6823 = 13646
```

Безусловно, проверку переменных на совпадение, можно было сделать одной, очень длинной строкой, но это запутывает как написание программы, так и её проверку, если что-то пойдёт не так. Программа должна быть читабельной, выражаясь современным языком!

Ну а теперь, как обычно, пора проделать упражнения!

Упражнение 7.7

Задачи сегодня несложные – надо решить простой числовой ребус:

```
  ЛИК
x ЛИК
БУБЛИК
```

с оператором умножения.

И ещё числовой ребус:

```
  РЕШИ
+ ЕСЛИ
СИЛЕН
```

Причём наибольшая цифра в слове "СИЛЕН" (это не ошибочное написание химического элемента "селен", а упрощённое написание краткого прилагательного "силён") равна 5.

А теперь настало время рассказать вам **ВСЮ правду про циклы**. Раньше я не мог этого сделать – вы были к этому не готовы, а сейчас, надеюсь, вы можете ЭТО выдержать. Дело в том, что структура операторов цикла сложнее, чем я вам рассказывал, и выглядит так:

```
while < логическое выражение >:
      < тело цикла >
else:
      <блок операторов, которые будут выполнены, если в теле цикла
не был выполнен оператор break>
```

Давайте напишем программу-мечту. Все математики XIX века мечтали о создании программы, вычисляющей простые числа (то есть числа, которые делятся нацело только на 1 и на само себя).

А в Питоне такую программу может написать любой начинающий программист, знающий оператор **break:**

```
N=100
for Y in range(2, N):
    x = Y // 2
    while x> 1:
        if Y % x == 0:
            break
        x -= 1
    else:
        print (Y, end=' ')
```

Опять же разберём программу. Будем искать простые числа в диапазоне от 2 до 99 (разумеется, вы можете присвоить **N** число 1 000 000, потом не спеша попить чайку – и только тогда увидеть завершение программы – приведённый в программе способ расчёта простых чисел довольно нетороплив), поэтому задаём **N** равным 100. Проверяем, делится ли число **Y** на **x** нацело, начиная с **x**, равному половине от **Y** (на числа, больше половины от **Y**, делить бессмысленно – всё равно нацело не разделишь, не правда ли?).

Допустим, **Y** разделился на **x** нацело – это значит, что у **Y** нашёлся делитель, отличный от 1, поэтому **Y** число составное и оператором **break** цикл **while** прерывается, и в цикле **for** берём следующее **Y**. А вот если, помаленьку уменьшая **x**, мы добрались до **x** = 1 – тогда цикл **while** прерывается без оператора **break**. Это означает, что у числа **Y** не найдено никаких делителей, следовательно, это число простое и его надо вывести на печать.

Да, чуть не забыл!

В цикле **for** тоже можно ставить оператор **else**, и работает он точно также, как и в цикле **while**:

```
for < переменная > in < последовательность > :
    < тело цикла >
else:
    < блок операторов, которые будут выполнены, если в теле
      цикла не был выполнен оператор break>
```

В связи с этим вот вам упражнение:

Упражнение 7.8

Давайте напишем несложную программу, которая в громадной последовательности находит какого-нибудь Айвазов…ой, это я немного вперёд забежал…Пусть найдёт число 154856 в последовательности, которая начинается с числа 101 и с шагом 171 перебирает все числа вплоть до 1213270. Если число нашлось – программа печатает "Список содержит число 154856", не нашлось –соответствующую запись. По завершении работы печатает, естественно, "Hasta la vista"

Ну а теперь – внимание, **смертельный номер**! **Задача из ЕГЭ**! До этого мы всё время делали программы, которые обрабатывали небольшие массивы чисел. Пришла пора заняться делом: найти нужные числа в огромных массивах.

Упражнение 7.9

Найдите все натуральные числа N, принадлежащие отрезку [200 000 000; 400 000 000], которые можно представить в виде $N = 2^m \cdot 3^n$, где m – чётное число, n – нечётное число.

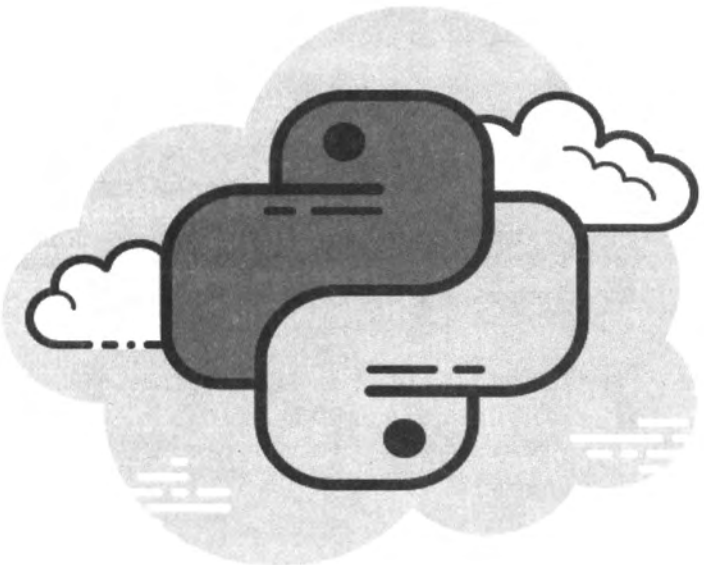
Циклы – наиважнейшее понятие в языках программирования. Позволяют повторять заданные действия почти до бесконечности. В Питоне бывают двух видов: циклы **for** выполняются строго определённое количество раз, а вот циклы **while** выполняются, пока выполняется заданное условие. Оператор **continue** заставляет Питон всё бросить и продолжить выполнение цикла с начала, а вот оператор **break** так же заставляет Питон всё бросить, но, наоборот, выйти из цикла.

А вот оператор **else** заставляет Питон проделывать какие-то дополнительные дела, если ни разу не сработал оператор **break**.

Ну, а теперь перейдём к ещё одному важному разделу в современном программировании – это **комментарии**!

Глава 8.

КОММЕНТАРИИ



Да, современные программы необыкновенно сложны, громоздки, а, главное, над одним проектом работает множество людей. Как объяснить другим людям, что вы хотели сделать в программе и каким образом?

Для этого и служат *комментарии.*

Всё, что идёт после знака # (по-русски это называется *решётка*, а на других языках – хэш, шарп, и даже, упаси Господи, октоторп), Питон не воспринимает как программный код вплоть до новой строки.

Например, вот так:

```
Std_Qnt += 1     # Добавился новый студент
```

Допустим, от предыдущего программиста вам достался вот такой фрагмент. Что он делает? Какой результат выдаёт? У кого спросить?

```
cnt=0
for x in range(1,1000):
    for y in range(x,1000):
        z=(x**2+y**2)**0.5
        if z==int(z):
            cnt+=1
            if cnt==144:
                print (cnt,'тройка Пифагора:', x,y,int(z))
print(cnt)
```

А вот так уже лучше и понятней:

```
# разработал Иванов С.С. для "Игры в математику" фирмы "Кукареку"
# Программа рассчитывает тройки Пифагора,
# т.е. находит целые числа, удовлетворяющие соотношению:
# x**2+y**2=z**2
# в данном фрагменте находит тройку № 144

cnt=0 # счетчик
for x in range(1,1000): #цикл для x
    for y in range(x,1000): #цикл для y
          z=(x**2+y**2)**0.5 #вычисляем z
        if z==int(z): # проверяем z на целочисленность
            cnt+=1 # тройка найдена! Счётчик +1
            if cnt==144:#нашли, например, 144-ую тройку
                print (cnt,'тройка Пифагора:',x,y,int(z))
#печатаем её
```

Комментарии, не побоюсь этого слова – важнейшая часть современного программирования. Комментируйте свои программы, не надейтесь на память – и будет вам счастье.

Короче, как "сахар придаёт неприятный вкус кофе – если забыть его туда положить"[1], так и комментарии в программе придают программе вкус и аромат.

О! Коли речь зашла о вкусе и цвете – самое время поговорить о кастрюле, в которой готовятся программы для Питона, то есть узнать, что ещё умеет **оболочка IDLE**.

1 Это из польского журнала «Пшекруй». Одно время в нем печатались вот такие смешные и мудрые «Мысли людей великих, средних и песика Фафика». Найдите в Интернете – не пожалеете!

Глава 9.

ОБОЛОЧКА IDLE

По чудесному совпадению, хотя ***оболочка IDLE*** и названа в честь одного из создателей, тем не менее она расшифровывается как *Integrated Development and Learning Environment* — то есть интегрированная среда для разработки и обучения. На самом деле это текстовый редактор с подсветкой синтаксиса, автозаполнением, умным отступом и другими функциями для удобства написания кода на Python.

Вы ведь заметили, что при написании программы в IDLE разные слова печатаются разными цветами?

- ***Операторы*** – **`import`**, **`def`**, **`while`** – *оранжевые*,
- ***Функции*** – **`str()`**, **`range()`**, **`print()`** – *фиолетовые*,
- ***Названия пользовательских функций*** и то, что выводится на экран – *синие*,
- ***Комментарии*** – *красные* (я же говорил, что это самое главное в современном программировании? Вот их и выделили ярче всех),
- ***Строки*** – *зелёные*,
- Все наши ***переменные*** – *чёрные*.

«Ну что ж, культурная картинка в целом получается»[1].

1 Отрывок из песни Тимура Шаова - поэта, сатирика, автора и исполнителя своих песен

Сначала, конечно, вы этого разноцветья не замечаете, но со временем это начинает здорово помогать: напечатали, например, **Print** – а слово остаётся чёрным. Непорядок!

Ах да, я же набрал **Print** с большой буквы! Исправляем на **`print`**, вспыхивает фиолетовая подсветка – порядок!

Или наоборот, решили вы переменную назвать коротко и ясно: TimeoutError, а она вдруг поменяла цвет на фиолетовый – это встроенный идентификатор, оказывается! Его нельзя применять в качестве имени переменной.

Ну что ж, исправим на TimeOutError – цвет изменился на чёрный – отлично!

Но! Как всегда – «Я художник, я так вижу!»[2] – ну не нравятся мне эти заданные исходные цвета. Нет проблем! Заходим в строку меню IDLE, нажимаем вкладку «**Options**» , затем в выпадающем меню нажимаем «**Configure IDLE**» – открывается окно «**Settings**» (Установки), открываем вкладку «**Highlights**» (Подсветки) – смотри рис. 7.

Сначала идёт «**Choose color for:**» (Выбери цвет для:), а ниже типы записей, которым меняем цвет – «**Python Definition**», «**Python Comments**» и так далее. Меняйте цвет, как вам заблагорассудится, потом нажмите «**OK**».

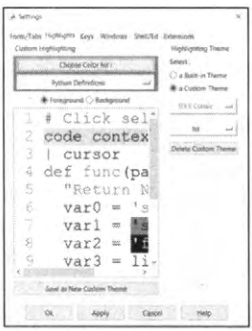


Рисунок 7

А ещё в окне «**Settings**» во вкладке «**Fonts/Tabs**» (Шрифты / Табуляции) можно выбрать шрифт, которым будут набираться программы, и его размер. А вот про табуляции здесь почему-то ничего нет…

2 Удивительно, но первым сказал эту фразу итальянский художник Паоло Веронезе ещё в XVI веке!

Умный отступ – это очень удобная штука в IDLE.

Все блоки в Питоне выделяются равным числом пробелов (4 штуки по умолчанию), а создаются они (блоки) операторами **`if`**, **`else`**, **`for`**, **`while`**, **`def`** после простановки двоеточия и нажатия «Enter».

Вот тут-то IDLE переходит на следующую строку и услужливо добавляет 4 пробела, что, согласитесь, довольно удобно.

А если мне надо уменьшить число пробелов, ну например – блок закончился?

Нажмите «Backspace» или стрелку «Влево». Если вы напечатали выражение **`return`**, **`break`** или **`continue`**, то указатель автоматически вернётся к прежнему отступу.

А ещё IDLE умеет подсказывать (а вот интересно, ябедничать своим создателям он умеет?).

Как только вы напечатали название функции или метода, то после открывающейся скобки (где далее будут прописаны аргументы) IDLE отобразит подсказку.

В ней будут описаны аргументы, которые ожидает функция (рис. 8) и подсказка, сообщающая, что эта функция делает.

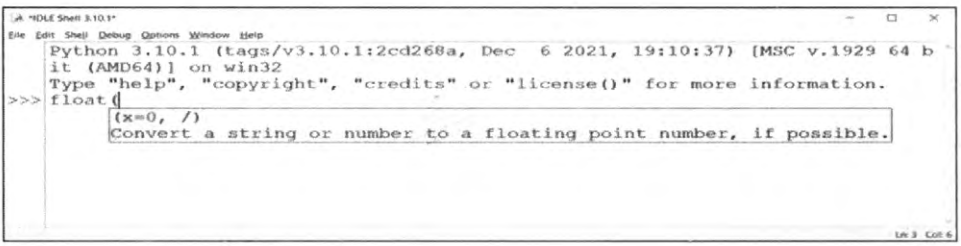


Рисунок 8

В частности, для функции **`float()`** IDLE подсказывает, что эта функция преобразует строку или число в число с плавающей запятой, если это воз-

можно (ну конечно, слово «балбес» довольно трудно превратить в число). А вот для метода **`random.randint()`** ... Я знаю, что это мы ещё не проходили – просто пример очень хороший!

Немного позже расскажу и про методы, и про модуль **random**.

Так вот, для метода **`random.randint()`** появляется подсказка **«(a,b) Return random integer in range [a,b] , including both end points»** (возвращает случайное целое число из промежутка [a,b], включая обе конечные точки).

Как видите, подсказки лаконичны и достаточно точны, надо только английский неплохо знать.

А ещё IDLE выдаёт историю команд. Как? Установите курсор после >>> и нажмите комбинацию «Alt+n», если надо полистать вперёд, или «Alt+p»для листания назад. А вот если надо скопировать какую-либо команду, поставьте курсор в её конец и нажмите «Enter» – между прочим, очень удобная возможность!

Во вкладке **File** находится обычный набор команд для работы с файлами: **Open, Save, Save as (Открыть, Сохранить, Сохранить как);** а вот команда **Recent Files (Недавние Файлы)** представляется мне очень полезной – она выдаёт список файлов, с которыми вы недавно работали.

В режиме редактирования файлов есть часто используемая вкладка **Edit.**

В ней находятся, пожалуй, общеизвестные команды, расскажу о некоторых.

- **Undo** – отменяет последнее изменение текущего окна, но в глубину не больше, чем 1000 раз (обидно, да? Что так мало-то!).
- **Redo** – отменяет отмену последнего изменения.
- **Cut** – вырезает выделенное в память системы, затем выделенное можно вставить, где скажите.

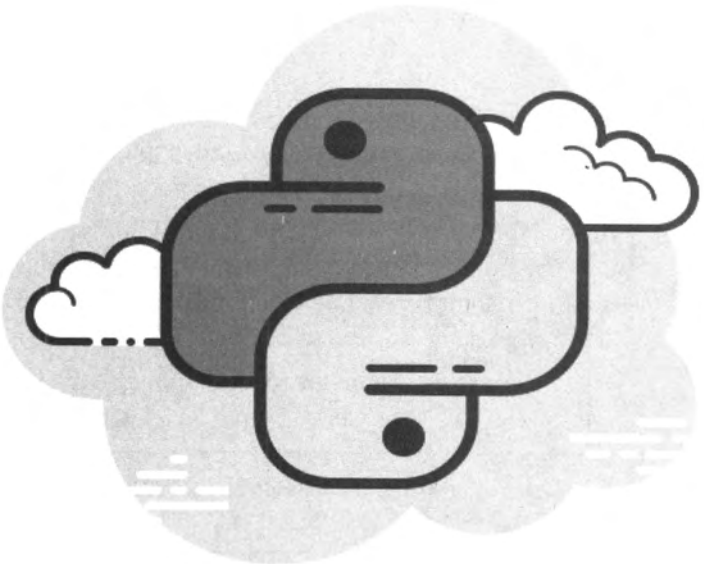
- **Copy** – копирует выделенное в память системы, затем выделенное можно вставить, где скажите.
- **Paste** – вставляет содержимое памяти системы туда, где стоит курсор, причём даже скопированное в других программах и Интернете. Именно так я вставил фразу на китайском в упр. 8.6.
- **Find** – находит, что попросите в вашей программе. Если программа большая, то поиск нужной команды/переменной становится тяжёлой задачей.
- **Find Again** – повторяет поиск.
- **Replace** – заменяет заданное выражение на другое заданное.
- В режиме редактирования файлов есть ещё одна очень полезная вкладка **Format.** Рассмотрим, что полезного можно найти в выпадающем меню.
- **IndentRegion** – смещает выбранные строки по ширине отступа (по умолчанию 4 пробела).
- **DedentRegion** – убирает отступ в выбранных строках (те же 4 пробела).
- **CommentOutRegion** – вставляет ## перед выбранными строками, то есть объявляет группу команд комментарием и Питон их теперь не выполняет.
- **UncommentRegion** – наоборот, удаляет первые # или ## из выбранных строк, то есть заставляет Питон выполнять группу команд, которые ранее были спрятаны как комментарии.

Оболочка-то IDLE оказалась не такой простой, как казалось сначала: она умеет подсвечивать разными цветами разные команды, в ней можно подстроить цвета и шрифт под ваши предпочтения, IDLE помнит названия последних написанных программ, подсказывает, что делает та или иная функция/метод…

В принципе, минимум программирования мы освоили. Ну почти. Впереди очередная важнейшая часть – **функции**.

Глава 10.

ФУНКЦИИ



В Питоне имеется некоторое количество так называемых ***встроенных функций***, то есть функций, к которым можно обращаться в любой момент, не совершая дополнительных заклинаний.

Это самые простые математические функции. Я приведу здесь список самых необходимых для дальнейшей жизни:

Функция	Описание
`abs(x)`	Возвращает модуль заданного числа: `abs(2)=2; abs(-3)=3`
`float(x)`	Преобразует целое число или строку в вещественное число: `float(7)=7.0; float('4.4')=4.4`
`int(x)`	Преобразует *x* в целое число, просто отбрасывая дробную часть: `int(9.99999)=9`
`max(x)`	Определяет самое большое число в списке, заданном через запятую: `max(1, 11, 5.5, 10)=11`
`min(x)`	Определяет самое меньшее число в списке, заданном через запятую: `min (1, 11, 5.5, 10)=1`

`pow(x,y)`	Возводит *x* в степень y: `pow(2, 5)=32`
`round(x)`	Округляет число до ближайшего целого по правилам математического округления: `round (2.5)=3;` `round (2.49999)=2`
`sum(x)`	Возвращает сумму значений последовательности: `sum(1,-1,2)=2`

Работают эти функции очень просто, например, пишем прямо в IDLE (или в самой программе):

```
>>>pow(5,500)
```

(это 5 в 500-ой степени) и мгновенно(!) получаем ответ:

```
30549363634996046820519793932136176997894027405723266638936139092812916
26524720457701857235108015228256875152693590467155317853427804283969735
13311420091788963072442053377285222035588819531883700816508667930179487
91366338993705251636497892270212003524508209121908744820211960149463721
10934030798550767828365183620409339937395998276770114898681640625
```

Да уж... Производительность компьютера на Питоне поражает...

С функциями `int()` и `float()` мы уже знакомы, с другими будем знакомиться на ходу:

Упражнение 10.1

Хитрая задача: придумайте и напишите программу, которая сообщает нам, пройдёт ли ящик размером A x B x C **в дверь размером** H x M. **В чём здесь хитрость? Надо обязательно использовать функции max и min.**

А вот синусов и всяких логарифмов в списке встроенных функций почему-то нет… Почему? Эти функции надо дополнительно подгружать из так называемого модуля **math** (об этом немного позже). Вообще, логику создателей питоновых (питонистых? питончатых?) модулей иногда понять довольно трудно…

Упражнение 10.2

Нехитрая задача: напишите программу, которая вычисляет корень квадратный из 2 и выводит результат построчно с разной, всё время возрастающей точностью.

Стоп! А как извлечь квадратный корень? Стандартной функции извлечения квадратного корня я что-то в списке не наблюдаю…

В Питоне почему-то многие функции *дублируют друг друга*. Например, квадратный корень можно извлечь стандартной функцией возведения в степень (квадратный корень = возведение в степень 0,5) или только что упомянутой функцией pow(x,y).

Хорошо, а как выводить разное число знаков после запятой? Ведь round() округляет всё до целого числа?

Как известно: «*Если ничто другое не помогает, прочтите же, наконец, инструкцию*»[1].

Читаем инструкцию к функции round() – второй, необязательный, параметр задаёт число знаков после точки:

```
>>>round (1.23456789,3)
1.235
>>>round (1.23456789,6)
1.234568
```

И даже

```
>>>round (832,-2)
800
```

Всё, тишина! Пишем программу!

1 Аксиома Кана из законов Мерфи. Что такое законы Мерфи? Систематизированное изложение законов подлости

Ну а теперь допустим, что в 3 (ещё хорошо, что в трёх! А если в 133?) частях программы нам надо выполнить совершенно одинаковые действия с разными числами. Скопировать один и тот же код 3 раза?! Ну это же не эстетично, и трудозатратно, особенно если код содержит в себе 100 операторов. Кроме того, это не дёшево и не практично…

В таких случаях помогают пользовательские функции, то есть создаваемые самим программистом.

Это просто:

```
def Fun(x1, x2, ...,xN):
    < блок >
    return A
```

Сначала мы оператором **def** (от английского *definition – определение*) создаём название функции (это может быть любая переменная), в скобках ставим параметры функции (в любом количестве), а после двоеточия, на следующих строках с отступом в 4 пробела! (это обязательно!) идёт *тело функции* – всё, что функция должна сделать.

Оператор **return** («возврат») присваивает *результат имени* функции и тут же «уходит» из неё. Функции следует определять *перед началом* текста основной программы.

Ну вот допустим, нам надо вычислить… Э… Давайте вычислим объём вытянутого кубика, «сокращённо» именуемого *прямоугольным параллелепипедом*. Перемножаем три стороны и получаем объём.

```
def Vol (x, y, z):# функция вычисления объёма помещения в куб.м
    return x*y*z

Home1 = Vol (5, 4, 3)
ExtraHome = Vol (11, 15, 2.8)
print (Home1)
print (ExtraHome)
```

Результат:

```
60
453.75
```

Я надеюсь, здесь всё понятно? В вышеприведённой программе мы дважды вызывали функцию **Vol()** с разными аргументами.

А что делать, если кроме объёма нам ещё и вычислить площади квартир Home1 и Extra? Вы догадались, что третий аргумент в функции **Vol()** – это высота потолка?

Можно сделать вот так:

```
def Vol (x, y, z):
    return x*y*z, x*y # объём и площадь квартиры

Extra = Vol (11, 15, 2.75)
print (Extra)
```

Получаем:

```
(453.75, 165)
```

В Питоне можно вернуть из функции сколько угодно переменных – да хоть 100!, перечислив их через запятую после команды **return**; полученная последовательность называется «кортеж» и записывается в круглых скобках.

Подробнее о кортежах я расскажу немного позже, в главе 16.

Упражнение 10.3

Напишите программу, рассчитывающую вес шайбы толщиной 1 мм. Вводим наружный диаметр, затем внутренний и программа выводит её вес. В программе надо применить функцию вычисления площади круга. Что? Из чего сделана шайба? Ну, допустим, из латуни. Удельный вес латуни марки ЛО62-1 равен 8,5 г/см^3.

Упражнение 10.3.1

Как убедиться, что программа работает верно? Придумайте некое тестовое задание для проверки.

Использованная в этом упражнении функция получилась очень простой – из одной строчки. Создавать отдельную функцию для такой мелочи неудобно, не так ли? Как обычно, всё уже предусмотрено заранее!

Для таких случаев в Питоне имеются так называемые **лямбда-функции**.

Вот, допустим, вам позарез потребовалось узнать, сколько литров воды в вашей дачной стандартной металлической 200-литровой бочке с размерами по ГОСТу 13950-91 – то есть диаметром 56,4 см и высотой 85 см; вот только воды в ней осталось только 55 см.

Формулу расчёта объёма цилиндра помните?

$$V = \frac{\pi D^2}{4} h$$

Вместо D берём 56,4 см, h – это высота воды в бочке в сантиметрах. Ну а если мы всё меряем в сантиметрах, то и результат получим в куб. см. А мне нужен результат в литрах (в куб. см задаётся объём иной жидкости, употребляемой отнюдь не для полива), значит, надо результат разделить на 1000 (1 литр это 1000 куб. см.).

А вот теперь определим лямбда-функцию, это делается очень просто:

```
X=lambda <аргументы>: <выражение>
```

Сначала пишем **lambda**, потом, через запятую, перечисляем аргументы (хоть сколько!), а после двоеточия пишем выражение, которое надо вычислить; полученный результат присваивается переменной **X**, например, возведём число в квадрат:

```
>>> square = lambda a: a*a
>>> square(5)
25
```

Так что для вычисления объёма воды в бочке имеем:

```
barr=lambda h: 3.14*(56,4**2)/4*h/1000
print ('в бочке',round(barr(55)),'литров')
```

Запускаем программу, смотрим результат: **в бочке 137 литров** – вполне разумно!

«А если у моей бочки другой диаметр?», – спросите вы – «например, 72 см. И воды в ней на 80 см». «Это где ж вы такую бочку достали?», – отвечу я. «Сварили на заказ, на оборонном заводе, ещё во времена СССР. Из титана, между прочим!» – услышу я в ответ. Всё с вами ясно, сейчас такая бочка стоит миллион, наверное… Немного улучшим программу – недаром ведь в определении лямбда-функции указываются аргументы (во множественном числе).

Перепишем программу:

```
barr=lambda h,D: 3.14*(D**2)/4*h/1000
print ('в бочке', round(barr(80,72)),'литров')
```

и пожалуйста – **в бочке 326 литров** – объёмистая у вас бочка, однако.

А зачем вообще потребовалось изобретать велосипед в виде этой лямбда-функции? Отвечаю: часто встречается ситуация, когда в программе надо несколько раз вычислять одно и тоже простое выражение из одной строки. Вот тут-то лямбда-функция и пригодится: её писать немного проще, а значит – быстрее! И второе – определить лямбда-функцию можно в любом месте программы, в отличие от просто функции, которую определяют строго в начале программы.

Упражнение 10.4

Жила-была пирамида, с размерами основания примерно 150x150 метров и высотой 49 метров. А потом пришли злые люди и снесли ей верхушку, оставив площадку размером

22x22 метра на высоте 42 метра. Надо, используя лямбда-функцию, определить объём пирамиды.

Вот теперь, как мне кажется, вы кинулись в другую крайность: «А зачем нужны обычные функции? Всё ведь прекрасно считается через лямбда-функции?» Как вы, надеюсь, заметили, результат работы лямбда-функции должен рассчитываться одной строкой, одним выражением, возможно и очень длинным. А вот если надо для получения результата применять оператор **if**, а то и вовсе цикл? Вот тут от пользовательской функции никуда не деться.

Например, придумайте функцию, которая при получении чётного числа печатает «ЧЁТ», нечётного – «НЕЧЁТ». А если вдруг ни то, ни сё? Ну, не целое число? Тогда печатает «Свят, свят! Таких чисел не бывает!». Да ладно, шучу – печатает «Не целое число».

Программа очень простая:

```
def TwoOrNot(N):
```

Сначала проверим, является ли аргумент чётным числом:

```
    if N%2==0:
        print('ЧЁТ')
```

А вот теперь тонкий момент. Если мы сейчас проверим число на нечётность оператором N%2!=0, то у нас все числа, включая не целые, попадут в нечётные.

Значит, сначала убедимся, что число целое, тогда оно, конечно, нечётное:

```
    else:
        if int(N)==N:
            print('НЕЧЁТ')
```

А сюда мы придём с не целыми числами:

```
        else:
            print ('Это не целое число')
```

Итак, функцию написали, теперь сама программа:

```
N=2020
```

Опаньки, а как вызывать функцию? Мы ведь **TwoOrNot** ничего не присваиваем? Ничего страшного, можно вызвать функцию просто так, без присваивания:

```
TwoOrNot(N)
M=2021.000001
TwoOrNot(M)
Hare=2023
TwoOrNot(Hare)
```

и убедиться, что всё прекрасно работает, прочитав на экране:

```
ЧЁТ
Это не целое число
НЕЧЁТ
```

В функцию не обязательно передавать все параметры, она запросто может использовать какие-то значения из основной программы, например:

```
def F(a):
    B=Const - a
    A=(a**2)/2+Const
    return A
Const=39.5
print(F(3))
Const=-1
print(F(1.0))
```

и на экране:

```
44.0
-0.5
```

То есть внутри функции преспокойно используется значение **Const** на момент вызова функции – так можно!

А теперь пример обычной пользовательской функции для подсчёта у числа количества его делителей, не считая 1 и самого числа:

```
# функция рассчитывает число делителей числа
def DEL (x):
    cnt=0 #счётчик делителей сбрасываем в 0
    for i in range (2, x//2+1): #перебор возможных делителей
        if x%i==0: #i является делителем
            cnt+=1 #счётчик +1
    return cnt #по окончании цикла - возвращаемся из функции
for j in range(35,41): #найдём кол-во делителей у чисел от
35 до 40
    print (j, 'имеет ',DEL(j),'делителей') #выводим результат
```

А вот и результат:

```
35 имеет 2 делителей
36 имеет 7 делителей
37 имеет 0 делителей
38 имеет 2 делителей
39 имеет 2 делителей
40 имеет 6 делителей
```

А теперь решите аналогичную задачу:

Упражнение 10.5

Назовём максимальным делителем числа A максимальное число, на которое делится A, за исключением самого числа. Рассчитайте разность между максимальными делителями чисел 124 и 303.

«А не замахнуться ли нам на Вильяма, понимаете ли, нашего Шекспира?[2]».

Ну, на Шекспира в книге по программированию мы, пожалуй, замахиваться не будем. А вот на решение сложного уравнения методом деления пополам – будем!

Допустим, нам надо быстренько решить уравнение:

$$1.24 \cdot x^5 + 0.12689 \cdot x^2 - 100.45 \cdot x - 36.235 = 0$$

Задачу сильно облегчает то обстоятельство, что надо найти хоть какой-нибудь корень с невысокой точностью ±0.02, и сильно утяжеляет то, что это надо было сделать, как обычно, вчера.

Вот тут-то нам и пригодится метод деления пополам. Для начала найдём отрезок, на котором заданная функция принимает разные значения, а значит, принимает значение 0 где-то внутри этого отрезка. Как это сделать? Ну не вручную же! Для этого напишем небольшую программу.

Для начала оформим заданную функцию в виде пользовательской функции:

```
def Fun(x):
    return 1.24*x**5+0.12689*x**2-100.45*x-36.235
```

Левый конец отрезка пусть будет 0:

```
x0=0
```

Вычислим и запомним значение функции в заданной точке 0:

```
f0=Fun(x0)
```

Теперь будем искать с помощью цикла с шагом 1 значение **x1**, при котором функция **f1** изменит знак на противоположный к **f0**:

```
for x1 in range(1,100):
    f1=Fun(x1)
```

2 Цитата из нашего, понимаете ли, бессмертного кинофильма «Берегись автомобиля» (1966 г.). Означает поставить перед собой более высокую, сложную задачу

А как узнать, что в точке **x1** функция изменила знак? Да очень просто: надо перемножить **f1** и **f2**:

```
if f0*f1<0:
```

если результат – отрицательный – то это здорово! (прямо как тест на коронавирус сдавали).

Печатаем найденную парочку:

```
print (x0,x1)
break
```

Результат:

```
0 4
```

Ага! Это означает, что в точке 3 функция ещё была того же знака, что и в точке 0, а в точке 4 она изменила знак на противоположный! Поэтому отрезок, на котором функция меняет знак – от 3 до 4! Запомним это.

Метод деления пополам состоит в том, что теперь мы берём значение, лежащее аккурат посередине найденного отрезка:

x01=(x0+x1)/2

и смотрим на значение функции:

f01= Fun(x01)

в этой точке. Это значение в любом случае отличается от значения функции слева **f0** или справа **f1**.

Если значения функций **f0** и **f01** имеют разные знаки, то в качестве нового значения **x1** мы берём **x01**, то есть наш интервал уменьшился ровно в 2 раза (точка **x01** – середина отрезка **[x0; x1]**) – смотри рис.9.

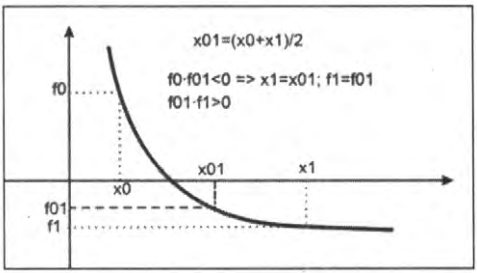


Рисунок 9

Понятно, что если **f0** и **f01** имеют одинаковые знаки, то надо изменить значение **x0** на **x01**!

Далее повторяем наш цикл до бесконечности… тпру, до какой ещё бесконечности? До достижения заданной точности! Вы ещё помните, до какой точности надо считать? До ±0.02! А что это значит?

Это значит, что разность значений **x0** и **x1** не должна превышать 0.04, тогда предлагаемое значение корня:

$$x=(x0+x1)/2$$

будет отличаться от точного значения меньше чем на 0.02 в ту или иную сторону.

Упражнение 10.6

Исходя из полученных теоретических знаний напишите программу, которая найдёт значение корня всё той же страшной функции

$1.24 \cdot x^5+0.12689 \cdot x^2-100.45 \cdot x-36.235=0$

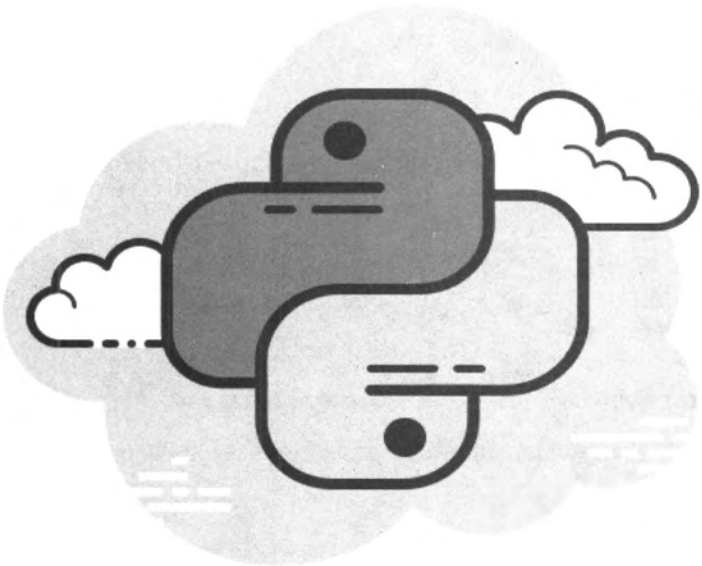
с точностью ±0.02 методом деления пополам. Помните, чему равен отрезок, на котором функция меняет знак? От 3 до 4!

Удивительно удобная это вещь – функция! Есть **встроенные функции** – для выполнения простых, наиболее часто встречающихся задач. А можно задать свою функцию – через оператор **def**. Написал функцию – и можешь её вызывать из самых неожиданных мест программы. **Название функции** – это как бы переменная, в него и попадает полученный результат.

Я надеюсь, что вас уже мучает вопрос, где же в Питоне спрятаны **математические функции**. Отвечаю – в главе 11!

Глава 11.

МАТЕМАТИЧЕСКИЕ ФУНКЦИИ



Вот так, запросто, вычислить в Питоне какой-нибудь sin(45°) не получится. Для начала надо импортировать модуль (то есть библиотеку) математических функций:

```
>>>import math
```

А вот теперь можно синус вычислить:

```
>>>Y=sin(90)
```

Не тут-то было – Питону не по душе такая прямолинейность! Программа выдаёт ошибку:

```
NameError: name 'sin' is not defined. Did you mean: 'bin'?
```

Перевожу: имя 'sin' не определено. Возможно, вы имели в виду: 'bin'?

Привыкайте, в Питоне обращаться к функции из модуля надо вежливо, через точечку:

```
>>>Y=math.sin(90)
>>>print (Y)
```

То есть сначала название модуля, а затем, через точку, название нужной функции.

На экране видим:

```
0.8939966636005579
```

А ведь синус 90⁰ равен 1! Так… Аргумент тригонометрических функций, видимо, не градусы…Ну конечно же, радианы! Хорошо, превратим 90 градусов в радианы – это π/2. Помните, чему равно число π? И я тоже не помню… Ничего страшного:

```
>>>Pi=math.pi
>>>Y=math.sin(Pi/2)
```

Ответ:

```
1.0 - гип-гип, ура!
```

А кстати, вот вам и число Эйлера, обожаемое математиками:

```
>>>E=math.e
>>>print (E)
2.718281828459045
```

Не очень хочется "таскать" с собой при каждом вызове название модуля **math**. Может, можно как-то попроще? Можно!

```
>>>from math import *
```

Эта команда означает: импортировать из модуля **math** все функции для обращения к ним обычным путём, без точки.

```
>>>q=cos(0)
>>>print (q)
```

На экране видим 1.0, как и положено.

Вам всё же удобнее работать с градусами? Пожалуйста:

```
>>>Grd=radians(90)
>>>print (sin(Grd))
```

Результат будет такой:

```
1.0
```

Вы же помните, что внутри оператора **print** можно выполнять различные вычисления?

Вот этим я иногда и пользуюсь.

Список математических функций, встроенных в модуль **math**, поистине необъятен (и включает такие странные функции, как **math.tau**, которая возвращает – вы будете смеяться – число 2π… Как говорится, "Умом Гвидо нам не понять…"[1]). Так что я приведу только небольшую часть наиболее нужных в обыденной жизни основных функций.

Функция	Описание
sqrt (x)	Возвращает корень квадратный положительного числа **x**
sin(x)	Синус числа **x**, аргумент в радианах
cos(x)	Косинус числа **x**, аргумент в радианах
tan(x)	Тангенс числа **x**, аргумент в радианах
asin(x)	Арксинус числа **x**, значение возвращается в радианах

1 Создатель языка Python — нидерландский программист Гвидо ван Россум. Ну, а автор, как обычно, позволил себе переделать знаменитое стихотворение Ф. Тютчева: "Умом Россию не понять, Аршином общим не измерить"

`acos(x)`	Арккосинус числа **x**, значение возвращается в радианах
`atan(x)`	Арктангенс числа **x**, значение возвращается в радианах
`log(x)`	Натуральный логарифм положительного числа **x**
`log10(x)`	Десятичный логарифм положительного числа **x**
`exp(x)`	Экспоненциальная функция e^x
`ceil(x)`	Округляет число "вверх", `ceil(2.0001)=3` (ceiling – с английского "потолок")
`floor(x)`	Округляет число "вниз", `floor(2.9999)=2` (floor – с английского "пол")
`radians(x)`	Переводит градусы `(x)` в радианы
`degrees (x)`	Переводит радианы `(x)` в градусы
`fmod(x,y)`	Возвращает остаток от деления **x** на **y** и эквивалентно `x%y`

Кстати, если вам нужен только один оператор из модуля **math**, можно (и нужно!) импортировать только его:

```
# Программа вычисления гипотенузы по заданным катетам
from math import sqrt #надо только квадратный корень
Cat1=float(input ("катет 1: "))
Cat2=float(input ("катет 2: "))
Hip=sqrt(Cat1**2+Cat2**2) # расчёт гипотенузы
print ("Гипотенуза равна", Hip)
```

Вы будете смеяться, но в модуле **math** имеется функция **hypot(x,y)**, которая вычисляет гипотенузу прямоугольного треугольника со сторонами **x** и **y**!

Упражнение 11.1

Ну что ж, коль у нас есть в руках такая куча математических операторов, просто грех этим не воспользоваться! Напишу-ка я программу для вычисления логарифма числа с заданным основанием. Стоп, а где мне взять функцию для логарифма с заданным основанием? Хм, попробую прочитать инструкцию к функции log(x) (где прочитать инструкцию? В интернете, конечно). Ага! Сначала пишем число, а потом, через запятую, основание логарифма.

Вводим сначала основание, а затем число, от которого желаем взять логарифм, после чего печатаем результат.

Упражнение 11.2

А вот теперь напишем программу, которая по 3 сторонам треугольника рассчитывает его углы. Увы, такого рода функции я в модуле math **не нашёл – всё же эта библиотека не бесконечна. Зачем вы взяли транспортир, циркуль и линейку? Ах, вы не помните, как посчитать углы и намерены их измерить? Ну, так я же вам подскажу – по теореме косинусов:**

$a^2=b^2+c^2-2bc\cdot\cos\alpha$

где a, b, c **– длины сторон треугольника, а** α **– угол, лежащий напротив стороны** a. **Отсюда вот вам и формула для расчёта угла:**

$\cos\alpha = (b^2+c^2-a^2)/(2bc)$

Сначала запрашиваем у пользователя длины всех сторон, вычисление угла оформляем в виде функции – её придётся вызывать трижды. Потом печатаем результат в градусах, естественно, мы же с вами не совсем математики, всё же. Да, и найденный угол выводим с точностью до 1 знака после запятой!

Помните, как выглядят КБ XX века? Сидят конструкторы

в белых халатах перед кульманами с листами ватмана и делают расчёты на странных калькуляторах, похожих на линейки. А этот калькулятор середины XX века (речь идёт о логарифмической линейке) выдавал результаты с точностью до 3 знаков, не больше[2]! Этого вполне хватает для технических целей.

Вот и нам негоже бахвалится питонистой точностью и размахивать 16-ю знаками после запятой. Скромнее надо быть.

А вот для расчёта атомной бомбы как раз и потребовались компьютеры с сумасшедшей точностью – процессы при взрыве идут за миллионные доли секунды, и ошибка в каком-нибудь 6 знаке после запятой приведёт к тому, что бомба не взорвётся.

Как, вы ещё не написали программу? Заболтались мы с вами, однако...

Да, и не забудьте придумать проверочные задачи. Какие? Простые! У меня их есть целых 3 штуки.

А почему в таблице нет генератора случайных чисел? Как Питон будет раздавать карты в программе, допустим, для игры в преферанс? За изготовление случайных чисел в Питоне отвечает отдельный модуль, причём небольшой. Итак, загружаем модуль **random**:

```
>>>import random
```

Привычным движением – через точку – вызываем метод генерации случайных чисел **random** (да, да, модуль и его функция называется одинаково... Я же говорю, с логикой проблемы)

```
>>>print (random.random())
```

2 КБ – конструкторское бюро. Кульман – устройство в виде наклонной доски с пантографом. Пантограф – устройство для проведения параллельных линий на чертеже. Ватман – хорошая чертёжная бумага, похожа на ту, на которой мы рисовали в школе, только поплотнее. Логарифмическая линейка – смотри Интернет, у меня нет слов для описания этого замечательного устройства

и получаем, например:

```
0.4805651874089585
```

При следующем запуске получим уже другое число:

```
>>>print (random.random())
0.27001258014532664
```

Метод **.random()** выдаёт случайное вещественное число от 0 до 1.

Можно, конечно, и так:

```
from random import *
O_o=random()
print (O_o)
```

Результат:


```
0.2123651183642557
```

Методов в модуле **random** немного – всего 22, в отличие от совершенно неопределённого количества в **math**. Перечислю самые нужные в домашнем хозяйстве:

Функция	Описание
`random()`	Возвращает случайное число из диапазона от 0 до 1
`shuffle(< список >)`	Перемешивает элементы списка случайным образом
`choice (< список >)`	Возвращает случайный элемент из заданного списка
`uniform (a,b)`	Возвращает случайное *вещественное* число **r** из диапазона **a<r<b**
`randint(a,b)`	Возвращает случайное *целое* число **r** из диапазона **a=<r=<b**

Немного непонятно? Примеры:

```
>>>import random
>>>print (random.uniform(0,5))
>>>print (random.randint(2,6))
```

На экране:

```
3.5370886281593155
2
```

А при следующем запуске увидим, например:

```
2.326510025895417
4
```

`.uniform (0,5)` выводит на экран *вещественное* число из диапазона (0; 5), а `.randint(2,6)` выводит *целое* число из диапазона [2; 6], то есть случайным образом выбранное число из списка [2,3,4,5,6].

И ещё примеры:

```
From random import *
Dau=['ши','ин','г ','да','фт','ун'] # задаём последовательность
shuffle(Sorry)# перемешаем элементы Sorry
print (Sorry)
```

Питон послушно перетасовал последовательность Sorry:

```
['ун','фт','ин','да','ши','г ']
```

Сколько раз надо запустить оператор **`shuffle()`**, чтобы составилось слово "дауншифтинг"[3], из которого я и собрал список **`Dau`**? По теории вероятности – 120 раз!

3 Дауншифтинг (от английского "сдвиг вниз") – отказ от принятых в обществе норм и переключение на "жизнь для себя"

Теперь добавим в программку пару операторов:

```
print (choise(Sorry))# выбираем случайный элемент из Sorry
print (choise(Sorry))# выбираем случайный элемент из Sorry
```

и... Изумлённо читаем:

```
File "C:\Users\4820992\AppData\Local\Programs\Python\Python310\trenik.py",
line 5, in <module>
    print (choise(Sorry))
NameError: name 'choise' is not defined. Did you mean: 'choice'?
```

(ОшибкаИмени: имя 'choise' не определено. Может, вы имели в виду: 'choice'?)

Какой же ты Питончик, умный! Ну конечно, я имел в виду 'choice'!

Видите, Питон неплохо подсказывает в случае появления ошибки. Как я и предупреждал выше, в программировании все названия операторов жёстко определены. Исправляемся:

```
print(choice(Sorry)) # выбираем случайный элемент из Sorry
print (choice(Sorry))# выбираем случайный элемент из Sorry
```

и читаем на экране, например:

```
'фт'
'ши'
```

То есть теперь Питон из перетасованной последовательности **Sorry** случайным образом выбирал элементы.

Упражнение 11.3

Простое упражнение. Напишите программу, которая будет "бросать" кубик (случайным образом выдавать числа от 1

до 6) при нажатии на любую клавишу. Подсказка: используем оператор *while*.

Стоп, а как вы собираетесь заканчивать игру? Привычно нажав Ctrl+C ?! Ни в коем случае!!! Ctrl+C используется только в крайнем случае, когда вы в процессе создания программы допустили ошибку и программа "зависла" – то есть вроде бы работает, но результатов никаких не выдаёт; в норме программа завершается сама или по нажатию заданной клавиши. Значит, надо сделать так, чтобы игра заканчивалась при нажатии на клавишу 'q'.

Упражнение 11.4

100 000 раз создайте случайное число методом .random(), сосчитайте сумму этих чисел и найдите среднее арифметическое. Догадайтесь, чему должно быть равно среднее арифметическое.

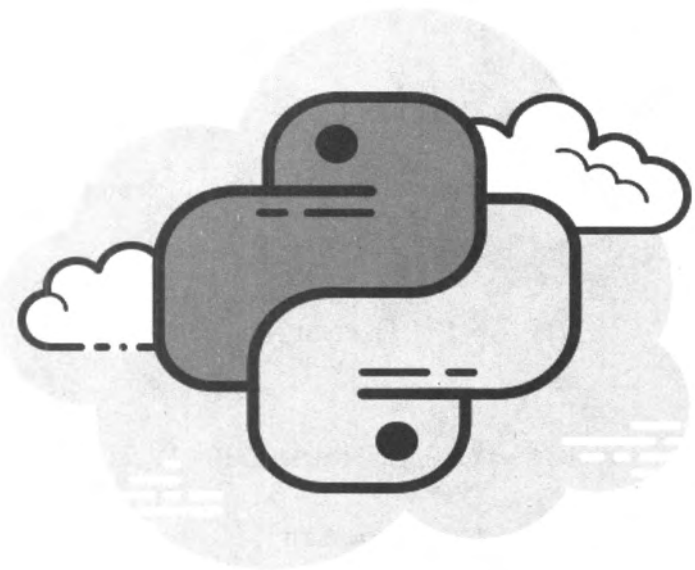
Для математических функций в Питоне предусмотрен отдельный закуток. Заклинание **import math** позволяет оживить сразу все функции из этого закутка. Обращаться с такими функциями надо осторожно, как со стеклянной вазой, через точечку. "Многоуважаемый логарифм **math.log (13.13)**, не соблаговолите ли сообщить нам натуральный логарифм числа 13,13"? Можно, конечно, и как обычно обратиться, после приказа **from math import ***: x=log(13.13). Или, если вы будете работать только с одним методом, можно вытащить только его: **from math import tan**. А ещё есть хранилище…? - ой, нет, – модуль! – **random**, который умеет раздавать карты (пока не видели, как это делается) и выдавать случайные числа.

Ну а если мы познакомились с функциями, то теперь сам Гильберт[4] велел узнать, что такое **рекурсия**.

4 Д. Гильберт – великий математик планеты Земля. Его как-то спросили об одном из его бывших учеников. "А, такой-то? — вспомнил Гильберт. — Он стал поэтом. Для математика у него было слишком мало воображения..."

Глава 12.

РЕКУРСИЯ



***Рекурсия** – это когда функция вызывает сама себя.*

"Как сама себя? Зачем? Ведь это же бесконечный цикл!" - вот такие мысли, конечно же, промелькнули у вас (автор безнадёжный оптимист) … Разумеется, функция вызывает сама себя каждый раз по-разному, пока этот процесс не прервётся каким-либо образом.

Классический пример – вычисление факториала:

$$F(N)= 1\cdot 2\cdot \ldots \cdot N$$

Обозначается N! (не надо кричать! Читается как "N факториал").

По-другому факториал можно определить как:

$$f(N) = N\cdot f(N-1), \text{ если } N \text{ не равно } 1;$$
$$f(1)=1$$

а это и есть рекурсия: f(N) определяется через f(N-1). Далее всё просто:

```
def fac(n): # факториал
    if n == 1:
        return 1 # факториал от 1 есть 1
    return fac(n-1) * n # рекурсия
```

```
Z=6 # число от которого находим факториал
x=fac(6) # вычисляем факториал через функцию
print(" ФАКТОРИАЛ" ,Z,'есть',x) # результат
```

Запускаем и имеем ответ:

ФАКТОРИАЛ 6 есть 720

Для чего нужен факториал? Ну вы даёте! Для перетасовки колоды карт, конечно! Первую карту из колоды в 36 карт можно достать 36 способами, вторую – 35 способами, … , последнюю – 1 способом. Итого колоду можно перетасовать **36! способами**.

Загружаем в IDLE вышеприведённую программу, исправляем **Z** на 36 и спокойно любуемся на ответ:

ФАКТОРИАЛ 36 есть 371993326789901217467999448150835200000000

А зря любуемся спокойно! Число-то грандиозное! Если всё население Земли – каждый человек каждую секунду – будет выдавать новый вариант перетасовки колоды – через сколько дней будут исчерпаны все варианты?

Полученный результат ошеломляет: для перебора всех вариантов перетасовки колоды из 36 карт надо 10^{24} (1 септиллион) лет. 10^{24} лет! Возраст Вселенной всего 10^{10} (10 миллиардов) лет! И это всем населением Земли, круглосуточно, без сна и отдыха, каждый житель планеты ежесекундно выдаёт новый вариант перетасовки! То есть мы только начали играть в карты!

Или: на данный момент создано всего-то примерно

0.000 000 000 000 000 000 000 000 000 002%

вариантов перетасовки…

Ну это упражнение, конечно, слишком простое и больше призвано вас удивить, точнее, поразить полученным результатом. А вот теперь задача немного позатейливее.

Упражнение 12.1

Напишите с помощью рекурсии программу, которая выводит на экран сумму вот такого знакопеременного ряда:

$$\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$$

Упражнение 12.2

Напишите с помощью рекурсии программу, которая выводит на экран сумму цифр натурального числа.

Упражнение 12.3

Используя рекурсию, создайте программу, которая находит наибольший общий делитель (НОД) двух чисел посредством алгоритма Эвклида. Как не знаете Эвклида?
Эвклид – великий учёный античности. Эвклид написал знаменитые "Начала", в которых изложил основы (начала) геометрии, на основе которых можно развивать геометрию строго логическим путём. Он создал основы теории чисел, доказал бесконечность множества простых чисел, придумал алгоритм нахождения НОД… О! Нам же это и надо!

Алгоритм Эвклида применяется к паре положительных чисел и на каждом шаге создаёт новую пару, которая состоит из меньшего числа и разности между большим и меньшим числом. Цикл повторяется, пока числа не станут равными, полученное число и есть искомый НОД исходной пары. Всё, осталось только написать программу. "Поверьте, что сможете, и полпути уже пройдено".[1]

1 Это сказал Теодор Рузвельт, 26-ой президент США.

Рекурсия – это просто когда функция вызывает сама себя, но уже с другими аргументами. Для чего это надо? Иной раз позволяет резко уменьшить длину программ.

Ну что ж, как я не раз уже намекал, пора переходить к любимому блюду Питона – это **строки**. Вот ими и займёмся в следующей главе.

Глава 13.

СТРОКИ

Строка – это просто последовательность символов, причём любых: латинских, английских, русских, татарских, корейских (там тоже буквы, а не иероглифы, как многие думают), китайских (см. упр. 7.6), а также цифр.

Как создавать строку, вы уже знаете:

```
>>>Vizbor='Давайте восклицать, друг другом восхищаться'[1]
```

Оп-па! В переменную имени Визбора попали слова Окуджавы! А в строке, как и в числе, содержимое в дальнейшем изменить нельзя! Вы ведь не можете в команде **pi= 3,1416927** ошибочно набранную цифру 6 вот так, запросто, как на бумаге, зачеркнуть на цифру 5? Придётся менять всё число:

```
pi=3.1415927
```

Так и со строчными переменными – изменить можно только если присвоить другое значение заново:

```
>>>Vizbor='Милая моя, Солнышко лесное.'[2]
```

Помните, что нумерация последовательностей в Питоне начинается с 0?

---

1 Автор – один из первых исполнителей бардовской песни – Булат Окуджава

2 Ю. Визбор. "Милая моя"

Набираем:

```
>>>Vizbor[0]
```

И видим:

```
'М'
```

То есть первая буква *значения* переменной **Vizbor** – "М"! А пятая?

```
>>>Vizbor[4]
'я'
```

Нумерация символов в последовательностях начинается с 0, поэтому пятый символ имеет 4-ый номер! Так, а теперь попробуем:

```
>>>Vizbor[-2]
'е'
```

Питон вернул нам предпоследний символ! Прелесть этой операции в том, что мне не надо знать длину строки, чтобы получать символы с конца.

А дальше начинается самое интересное.

Есть такая операция – *срез строки* (мы ей, честно говоря, сейчас и воспользовались).

Её формат:

[<начало>:<конец>:<шаг>]

Самое смешное, что все три параметра в ней – необязательные. Понятно, что срез с одним параметром возвращает один символ с соответствующим номером, что мы и проделывали выше.

Срез с двумя параметрами: **Vizbor[i:j]** вернёт подстроку из **j - i** символов, начиная с символа с индексом **i** и до символа с индексом **j**, *не включая* его.

Например, хочу символы с 11 по 18:

```
>>>Vizbor[11:19]
'Солнышко'
```

А теперь все символы, начиная с 6-го:

```
>>>Vizbor [6:]
'моя, Солнышко лесное.'
```

А вот если указать все 3 параметра, то:

```
>>>Vizbor[0::2]
'Млямя оншолсо.'
```

– можно вывести каждый второй символ,

```
>>>Vizbor[::-1]
'.еонсел окшынлоС ,яом яалиМ'
```

И даже перевернуть строку!

Но! Сама строка **Vizbor** при этом ни чуточку не меняется, то есть если вам надо выделить 1, 3 и 5 символ из строки и дальше с ним работать, надо присвоить это значение другой переменной:

```
>>>Viz1=Vizbor[0:6:2]
>>>Viz1
'Мля'
```

А как узнать, сколько символов в строке? Очень просто – функцией **len** (от английского *length* – "длина"):

```
>>>len(Vizbor)
27
```

Пробелы тоже считаются! Не правда ли, очень полезная функция? Теперь понятно, почему функция **range(100)** даёт последовательность от 0 до 99 включительно? Это удобно при работе с последовательностями. Вот для примера распечатаем строку … Строку? Да это же проще некуда:

```
>>>print (Vizbor)
Милая моя, Солнышко лесное.
```

А если с каждым элементом строки я хочу что-нибудь сделать? Придётся запускать цикл:

```
Lerm='Белеет парус одинокий'
for i in range(len(Lerm)):
    print(Lerm[i],' ',end='')
```

На экране:

```
Б е л е е т  п а р у с  о д и н о к и й [3]
```

Правда, как вы помните из главы 8, строку можно распечатать более элегантным способом:

```
Lerm='Белеет парус одинокий'
for i in Lerm:
    print(i,'.',end='')
```

```
Б .е .л .е .е .т .  .п .а .р .у .с .  .о .д .и .н .о .к .и .й .
```

Между прочим, строковые переменные можно складывать. И присваивать новое значение старой переменной. И таким образом, немного обманывать Питон, всё же меняя значение строковой переменной, так же, как и числовой:

3 А знаете ли вы, что фамилия Лермонтов – шотландского происхождения? И ведет своё начало от Джорджа Лермонта, шотландского наемного воина, осевшего в России в XVII веке

```
>>>Vizbor='Милая моя, Солнышко лесное,'
>>>Vizbor2='где, в каких краях встретишься со мною?'
>>>Vizbor2= Vizbor + Vizbor2
>>>Vizbor2
'Милая моя, Солнышко лесное, где, в каких краях встретишься со мною?'
```

С помощью этой хитрости и цикла по всей строке давайте уберём из переменной **Vizbor** все пробелы:

```
Vizbor='Милая моя, Солнышко лесное.'
L=len(Vizbor) # длина строки
W='' # создаём новую переменную
for i in range(L): # цикл по всей строке
      if Vizbor[i]!=' ': # если символ не пробел
            W=W+Vizbor[i] # то добавляем его в W
print (W) # печатаем результат
```

Программа напечатает:

Милаямоя,Солнышколесное.

Упражнение 13.1

С помощью очень полезной функции len() найдите пятое слово в текстовой переменной *Txt*, присвойте его переменной *Txt2* и напечатайте её. Чему равна переменная *Txt*? Ну экие вы беспомощные... пусть это будет стихотворение самого популярного в России поэта. Ну что вы, это не Пушкин... Этот стих легко процитирует практически любой россиянин в возрасте 3+, а вот со стихами Пушкина мы знакомимся только лет с 7...

Ну? Догадались?

"Наша Таня громко плачет: уронила в речку мячик[4]"

Подсказка: слова отделяются друг от друга пробелами.

4 А. Барто "Мячик"

Строки можно складывать в любом количестве и порядке, а операция сложения носит гордое название *конкатенация*:

```
>>>S1='абра'
>>>S2='кад'
>>>SS=S1+S2+S1
>>>SS
'абракадабра'
```

Оператор **in** позволяет узнать наличие подстроки в строке

```
>>>'Абр' in SS
False
```

В переменной SS все слова идут с маленькой буквы, поэтому Питон справедливо считает, что подстроки 'Абр' в слове "абракадабра" нет, а вот 'даб' – есть:

```
>>> 'даб' in SS
True
```

Все догадались, что **in** в переводе с английского "в"?

Ну и напоминаю, что оператор * позволяет повторить строку заданное число раз:

```
>>>print ("ку-ку "*10)
ку-ку ку-ку ку-ку ку-ку ку-ку ку-ку ку-ку ку-ку ку-ку ку-ку
```

А теперь немного отвлечёмся.

Как, я надеюсь, вы понимаете, буквы в компьютере представляются длиннющей чередой нулей и единиц, причём у каждой буквы – свой набор единичек и нулей.

А вот какой букве какой код соответствует – определяется таблицей Юникод, в которой содержатся коды для почти 1,1 миллиона (!) символов. Куда так много? А там содержатся символы для почти всех алфавитов мира и китайских иероглифов в придачу. Так что каждому символу соответствует свой номер в этой таблице, например " " (пробел) идёт под номером 40, буква 'щ' – под номером 1097…

А как узнать номер символа в таблице Юникод? Функцией **ord()**:

```
>>>ord('щ')
1097
>>>ord('Щ')
1065
>>>ord('4')
52
```

Да, **символ** цифры 4 отличается от цифры 4 (которая имеет код 00000010).

А как сделать наоборот – по номеру из таблицы Юникода напечатать символ? Запросто – функцией **chr()**:

```
>>>chr(1065)
'Щ'
>>>chr(12106)
'木'
```

Вот вам и код китайского иероглифа '木' – дерево.

А теперь используем полученные знания в не совсем мирных целях:

Упражнение 13.2

А не поиграть ли нам в разведчиков? Что? Надоело в разведчиков? Тогда в шпионов! Давайте взломаем сообщение, зашифрованное простейшим способом –

так называемым "шифром Цезаря"[5].. В этом шифре каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите. Например, в шифре со сдвигом вправо на 2, буква А была бы заменена на В, Б станет Г, В – Д и так далее. Число "2" в этом случае называется ключом.

Допустим, перехвачено сообщение от вражеского шпиона: "Утейимцйпґ$сй$хчиґц", созданное шифром Цезаря. Пользователь вводит предполагаемый ключ, программа производит обратный сдвиг в сообщении и выводит полученную абракадабру на экран. Если полученный результат нас удовлетворяет – печатаем "да" и прекращаем работу; если нет – печатаем что угодно и продолжаем игру в угадайку.

Разумеется, при решении задачи надо использовать функции ord() и chr().

Очень полезна функция **str()**, которая тупейшим образом превращает то, что стоит в скобках в строку:

```
>>>str(123)
'123'
>>>str(3.14)
'3.14'
```

И даже

```
>>>str(True)
'True'
```

5 Гай Юлий Цезарь (100 год до н. э. — 44 год до н. э.) — древнеримский государственный и политический деятель, писатель, полководец. От его фамилии, между прочим, происходит и русское слово "царь" (на древнерусском оно писалось как цьсарь (ь в древнерусском произносилась как краткий звук е)

Упражнение 13.3

Напишите программу, которая вводит целое число, а выводит последнюю цифру квадрата введённого числа; используем функцию str(). Кто сказал: "Ну это же просто!"? Хорошо, уточняю: программа должна занимать ОДНУ (да-да, одну) строку.

Упражнение 13.3.1

А теперь хочу напечатать первую цифру квадрата введённого числа.

Кроме вышеописанных функций работы со строками, существует ещё огромное количество методов работы со строками, которые преобразуют строку, *оставляя её неизменной.* Если нужен полученный результат – присвойте его другой переменной (а можно и той же самой). Как обычно, не очень понятно?

Тогда такой примерчик: восстановим переменную **Viz1**:

```
>>>Viz1='Мля'
```

Чтобы применить к объекту какой-нибудь метод, надо приписать его (метод) через точку. Применим метод **.swapcase()**:

```
>>>viZ1=Viz1.swapcase()
>>>print(viZ1,Viz1)
мЛЯ Мля
```

То есть метод **.swapcase()** заменил верхний регистр на нижний и наоборот, при этом переменная **Viz1** не изменилась, и нам пришлось результат загрузить в переменную **viZ1**!

Это и есть главный недостаток и главное преимущество строк: никакие методы не меняют содержимое обрабатываемой строки, а чтобы получить результат преобразования строки, его надо присвоить другой переменной (а можно и той же самой!).

Работу основных методов обработки строки будем познавать на примерах. Для примера возьмём переменную

```
>>>Aref = "я не русалка - меня сом недоглотил!"[6]
```

и будем применять к ней разные методы.

Метод `.capitalize()` делает прописной первую букву строки:

```
>>>Aref.capitalize()
'Я не русалка - меня сом недоглотил!'
```

Метод `.title()` делает прописной первую букву каждого слова:

```
>>>Aref.title()
'Я Не Русалка - Меня Сом Недоглотил!'
```

Ну теперь, как говорится,

"Читатель ждёт уж рифмы "розы"
На вот, возьми её скорей!"[7]

– сделаем все буквы прописными:

```
>>>Aref.upper()
'Я НЕ РУСАЛКА - МЕНЯ СОМ НЕДОГЛОТИЛ!'
```

6 О. Арефьева "Одностишийа" . Это не опечатка! Сборник интереснейших одностиший Ольги Арефьевой действительно так и называется

7 А.С. Пушкин "Евгений Онегин".

Побаловались – и хватит. Вернём всё как было:

```
>>>Aref.lower()
'я не русалка - меня сом недоглотил!'
```

Сделаем что-то вроде заголовка:

```
>>>Aref.center(50, '+')
'+++++++я не русалка - меня сом недоглотил!++++++++'
```

Данный метод, понятное дело, центрирует строку, заполняя излишки справа и слева знаками + (это у меня. А по умолчанию пробелами). Первый параметр определяет длину новой строки, а второй – каким символом будем заполнять.

Два аналогичных метода **.ljust(width, char)** и **.rjust(width, char)** тоже делают строку длиной **width** (ширина с английского) по необходимости заполняя последние/первые места символом **char**:

```
>>>Aref.ljust (38, '~')
'я не русалка - меня сом недоглотил!~~~'
>>>Aref.rjust (40, '@')
'@@@@@я не русалка - меня сом недоглотил!'
```

Ну а метод .strip() удалит указанные знаки в начале и в конце строки:

```
>>>Aref. strip('я')
' не русалка - меня сом недоглотил!'
>>>Aref. strip('!')
'я не русалка - меня сом недоглотил'
```

Для последующих опытов оставим в покое русалок и перейдём к поросятам:

```
>>>Pig='''О храбрый, храбрый пятачок!
Дрожал ли он? О нет! О нет!'''[8]
```

Строки, которые обрабатывает Питон, могут быть очень длинные или даже поэтические, как в нашем случае.

Для записи таких строк с сохранением форматирования и переноса строк используется *тройной апостроф* в начале и в конце строки.

Сколько раз в тексте встречается слово *нет*?

```
>>>Pig.count('нет')
2
```

А буква *р*?

```
>>>Pig.count('р')
5
```

А в какой позиции впервые встречается слово *нет*?

```
>>>Pig.find('нет')
44
```

А вот номер **последнего** вхождения находится так:

```
>>>Pig.rfind('нет')
51
```

То есть ищем вхождение в обратном (**reverse**) порядке – с конца строки, поэтому метод называется **rfind.**

8 А. Милн "Винни-Пух и все-все-все". Пересказ с английского Б. Заходера

А если подстрока не найдена?

```
>>>Pig.find ('да')
-1
```

Ага! Если строка не найдена, то мы получим -1 (такой же ответ даёт метод **.rfind()**). Ну и зачем всё это, спросите вы? Я и так могу сказать, входит слово в строку или нет... Ну, во-первых, программы обычно имеют дело со строками, состоящими из миллионов(!) символов. А во-вторых, вот вам простое

Упражнение 13.4

Составьте программу, которая считает, сколько раз во фразе

"Подумав немного, он положил на блины самый жирный кусок сёмги, кильку и сардинку, потом уж, млея и задыхаясь, свернул оба блина в трубку, с чувством выпил рюмку водки, крякнул, раскрыл рот...

Но тут его хватил апоплексический удар."[9]

встречается тот или иной набор букв, введённый пользователем. А вот здесь пусть выход из программы происходит, когда пользователь нажал любую цифру.

Подсказка: метод .isdigit() выдаёт True, если строка состоит из цифр.

Упражнение 13.5

А теперь хочу увидеть программу, которая определяет, является ли введённая строка палиндромом, то есть читается одинаково слева направо и справа налево.

Подсказка: а вот здесь надо применить метод .replace(old, new),который заменяет все вхождения подстроки *old* **на подстроку** *new*.

9 А. Чехов "О бренности"

Почему я не рассказал об этом сразу, в методах обработки строк? "Никто не обнимет необъятного[10]", а Питон совершенно необъятен ...

Что? Пример палиндрома?! Вы не читали в детстве про Буратино?! "А роза упала на лапу Азора" – это оттуда классический палиндром; но А. Толстой позаимствовал его у Афанасия Фета – знаменитого русского поэта.

Но давайте лучше используем мой собственный, авторский палиндром:

"И Вере наша Саша не реви..."

который я написал про своих детей :)

Остались ещё силы на упражнения?

Тогда надо написать полезную программу:

Упражнение 13.6

Обычное явление – когда по неизвестной причине в тексте почему-то появляются лишние пробелы, вот как сейчас у меня. Задача – написать простую программу, которая удаляет лишние пробелы в тексте.

А в качестве подопытной фразы используем начало весёлой повести Джерома К. Джерома :

"*Нас было четверо — Джордж, Уильям Сэмюэль Гаррис, я и Монморенси[11]***".**

Удалите лишние пробелы и сосчитайте улов (то есть сколько пробелов удалили)

10 Козьма Прутков

11 Да, повесть "Трое в лодки, не считая собаки" начинается именно так: "Нас было четверо"

Строка – это просто обычный текст, присвоенный любой переменной. Помимо очень эффективной операции *среза строки* существует ещё множество методов обработки строки.

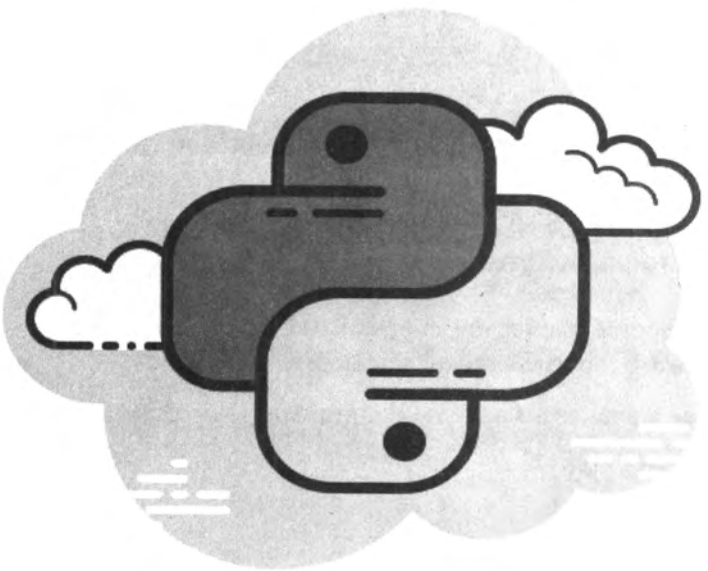
Однако здесь надо помнить, что строки в Питоне *неизменны*, так что если надо запомнить результаты работы над строкой – присвойте его какой-нибудь переменной.

Методов обработки строки аж 47 штук, самые нужные я перечислил в **Приложении 1**.

В общем, вы уже поняли, что список методов обработки строки поистине огромен, проще ознакомится с ним в Интернете, когда приспичит... Займёмся другим видом последовательности, который в Питоне называется **списки** (хотя мне лично больше нравится привычное название *массив*, но тут, как говорится, "поздняк метаться" – слово "список" стало привычным термином в Питоне).

Глава 14.

СПИСКИ



Список *– это тип данных, предназначенный для хранения набора или последовательности разных элементов.*

Как обычно, ясно, что ничего не ясно. Ладно, создадим список:

```
>>>Sps=[1, 2.05, 'абвг', [6.6, 120],"eight"]
```

Здесь **Sps** имя списка, а в квадратных скобках через запятую перечислено всё, что в него входит:

1. На позиции № 0 число 1
2. На позиции № 1 – число с плавающей запятой 2,05
3. Строка **абвг**
4. Список **[6.6, 120]**
5. Строка **eight** на позиции № 4.

То есть в качестве элементов списка может быть всё, что угодно – числа, строки и даже … другие списки, как видите.

Обращаются к элементам списка точно так же, как к строке:

```
>>>Sps[0]
1
>>>Sps[3]
[6.6, 120]
>>>Sps[-4]
2
```

Функция **len()**, как всегда, даёт длину списка (а именно – количество элементов):

```
>>>len(Sps)
5
```

А теперь давайте распечатаем список. Какой? Другой:

```
Digit=[1,'one',2,'deux',3, 'drei']
for i in range(len(Digit)):
    D=Digit[i]
    print (D, end=' ')
```

На экране, как и ожидалось:

```
1 one 2 deux 3 drei
```

Здесь в цикле меняется индекс элемента **i**, затем выводится элемент списка с индексом **i**.

Однако в Питоне есть более эффектный (и эффективный) способ распечатать весь список:

```
Digit=[1,'one',2,'deux',3, 'drei']
for j in Digit:
    print (j, end=' ')
```

Здесь элементы списка выводятся в одну строку, разделённые пробелом, при этом в цикле переменная ***j*** принимает сами значения из списка – точно так же, как в предыдущей главе мы распечатывали строку Лермонтова.

Данный способ удобен, когда нам надо пробежать последовательно все элементы списка. Если же надо как-то дополнительно обработать элементы списка, то придётся применить первый способ:

```
Digit=[1,'one',2,'deux',3, 'drei']
for i in range(0,len(Digit),2):
    D=Digit[i]
    E=Digit[i+1]
    print (D, E)
```

на экране видим:

```
1 one
2 deux
3 drei
```

Это более удобный способ показать число и его перевод на языки мира – английский, французский и немецкий, если вы ещё не догадались.

Операция *среза строки* точно так же работает и со списками:

```
>>>Sps[1:4]
[2, 'абвг', [6.6, 120]]
```

Списки относятся к *изменяемым* типам данных, в отличие от строк.

Например, заменим 4-ый элемент в Sps на **ω**:

```
>>>Sps[3]='ω'
>>>Sps
[1, 2, 'абвг', 'ω', 'eight']
```

Мы благополучно заменили 4-ый элемент списка!

А вот *добавить* элемент в список вот так запросто не получится:

```
>>>Sps[5]=2.12
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    Sps[5]=2.12
IndexError: list assignment index out of range
```

То есть "ИндексОшибка: индекс назначения списка вне диапазона".

Со всеми видами последовательностей в Питоне следует обращаться ответственно, используя только разрешённые Создателем[1] методы.

Добавлять элементы в список, конечно, можно, но осторожно. Для этого есть самый удобный метод **.append()**, добавляющий 1 элемент в *конец* списка:

```
>>>Dig1=[1,3,4,2,6,2,6]
>>>Dig1.append(2.12)
>>>Dig1
[1, 3, 4, 2, 6, 2, 6, 2.12]
```

Отлично! А теперь добавим в конец списка буковку:

```
>>>Dig1.append(F)
```

Оп-па! Что-то пошло не так, всё ж таки Питон не всеядный:

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    Dig1.append(F)
NameError: name 'F' is not defined
```

Примерный перевод: "Отслеживание последнего вызова", дальше указывается в каком файле и в какой строке произошла проблема и указывается

[1] Здесь под Создателем имеется в виду Гвидо ван Россум – основной изобретатель Питона

оператор, на котором споткнулся Питон; и в последней строке самое главное для нас – переменная **F** не определена!

Всё, что не в кавычках (кроме чисел, конечно) Питон считает переменными (вот поэтому имя переменной не может начинаться с цифры). Теперь понятно, как добавить буковку:

```
>>>Dig1.append('F')
>>>Dig1
[1, 3, 4, 2, 6, 2, 6, 2.2,'F']
```

А можно добавить содержимое переменной:

```
>>>a="Бамбарбия! Кергуду!"[2]
>>>Dig1.append(a)
>>>Dig1
[1, 3, 4, 2, 6, 2, 6, 2.2, ,'F','Бамбарбия! Кергуду!']
```

Порядок! Работает!

Упражнение 14.1

Дан некий список из чисел. Выведите все его чётные элементы. При этом используйте цикл *for*, перебирающий элементы списка, а не их индексы!

Упражнение 14.2

Дан некий список из 11 чисел. Создайте новый список, состоящий из средних арифметических значений всех его последовательных троек элементов, с точностью до 2 знаков после запятой.

А если мне надо добавить элементы не в конец, а любое место списка?

Желаете метод? Он есть у меня!

2 Надеюсь, вы все смотрели "Кавказскую пленницу"?!

Метод `.insert()` вставляет 1 элемент в указанную позицию:

```
>>>Dig1.insert(0, True)
>>>Dig1
[True, 1, 3, 4, 2, 6, 2, 6, 2.2, 'Бамбарбия! Кергуду!']
>>>Dig1.insert(2,chr(12103))
>>>Dig1
[True, 1, '日', 3, 4, 2, 6, 2, 6, 2.2, 'Бамбарбия! Кергуду!']
```

То есть, мы успешно вставили во 2-ю позицию списка китайский иероглиф "день, сутки" по его номеру в таблице Юникод.

Создать список из *строки* можно с помощью функции `list()`:

```
>>>Lst=list("Привет")
>>>Lst
['П', 'р', 'и', 'в', 'е', 'т']
```

А ещё можно списки складывать (по-умному это, конечно, называется конкатенация):

```
>>>Sps=Sps+["Пятый элемент"]
>>>Sps
[1, 2, 'абвг', 'ω', 'eight', 'Пятый элемент']
```

Для работы со списками есть всего-то 11 методов, причём они частично совпадают с методами работы со строками.

Например, создадим новый список (всё-таки списки, как правило, состоят из однородных элементов – или чисел, или букв, или строк…):

```
>>>Dig1=[1,2,3,4,2,6,2,6]
```

И сосчитаем, сколько раз в этом списке содержится число 2:

```
>>>Dig1.count(2)
3
```

Но большая часть методов, свои, оригинальные, списочные, например, метод **.index(X)** возвращает положение первого элемента с заданным значением:


```
>>>Dig1.index(2)
1
```

Не забыли, что нумерация элементов начинается с 0?

У метода **.index(x)** есть недостаток: если элемента **x** нет в заданном списке, то произойдёт ошибка и программа остановится…

Поэтому лучше сначала проверить наличие элемента в списке. Как? Либо с помощью метода **.count(x)**:

```
>>>Dig1.count(2)
3
```

Либо более простым способом – оператором **in**:

```
>>>2 in Dig1
True
>>>5 in Dig1
False
```

А вот метод **.pop(i)** удаляет из списка элемент с номером *i* и выводит его на экран. Если номер не указан, удаляет последний элемент:

```
>>>Dig1.pop(1)
2
```

Как видим, Питон удалил элемент с номером 1 и услужливо напечатал его.

Обратите внимание, что *методы списков*, в отличие от *строковых методов*, изменяют сам список, и поэтому результат не нужно записывать в этот список – он изменится сам:

```
>>>Dig1
[1, 3, 4, 2, 6, 2, 6]
```

А как из списка сделать строку? О, это не так просто! Дело в том, что функция str(), делающая строку из всего, что угодно, делает из списка что-то странное:

```
>>>vvv=['veni,','vidi,','vici']
>>>str(vvv)
"['veni,', 'vidi,', 'vici']"
>>>s=[1,2,3]
>>>str(s)
'[1, 2, 3]'
```

Поэтому "нормальные герои всегда идут в обход"[3]!

Во-первых, список должен состоять только из строчных элементов.

Во-вторых, собирает из этих элементов строку хитрож… да-да, всё правильно: хитро-жуткий метод `.join()`:

```
<строка>=<разделитель>.join(<список>)
```

Попробуем теперь сделать этим методом из списка **vvv** нормальную строку:

```
>>>W=' '.join(vvv)
>>>W
'veni, vidi, vici'
```

[4]

3 Фильм "Айболит-66" уже подзабыт, а эта фраза из него стала крылатой!

4 Переводится как "Пришёл, увидел, победил". Латинский афоризм – о быстрой и решительной победе

То есть метод `.join()` собирает из элементов списка строку, вставляя между элементами заданный разделитель – в данном случае пробел; но можно вставить и пустой разделитель, и любые другие знаки:

```
>>>W='+:'.join(vvv)
>>>W
'veni,+:vidi,+:vici'
```

Я чувствую, что вы немного засиделись. Предлагаю размяться – сделать упражнение!

Упражнение 14.3

Имеем строчку: "Значит, так:, автобусом, к, Тамбову, подъезжаем,"[5], в которую какие-то враги понаставили лишних запятых. Надо из строки сделать список, лишние запятые убрать (все, кроме последней), а исправленный список превратить назад в строку и затем вывести на печать.

Метод .join вам в помощь.

А теперь предлагаю поиграть в шпионов… Как недавно играли?! И правда, играли… Хорошо, поиграем опять в разведчиков – это вас устроит?

Упражнение 14.4

Это у них шпионы, а у нас – чистые, благородные э… добытчики чужих секретов. Допустим, разведчикам удалось приобрести на чёрном рынке совершенно секретную супер-пупер-ЭВМ, запрещённую к вывозу из вражьей страны. Тут как раз есть контейнер с 1000 ящиков, отправляемый в Россию. Нам надо случайным образом в один из ящиков загрузить эту ЭВМ, и сообщить на Родину, в каком из ящиков лежит образец.

Причём сообщить надо так, чтобы никто не догадался. Для

5 В. Высоцкий "Товарищи учёные"

этого создаём случайную последовательность из 1000 букв латинского алфавита. Так вот номер позиции, где впервые встретилось сочетание букв "au" и даёт нам номер ящика со спецтоваром. Почему "au"? Символ золота в таблице Менделеева – "Au"! А что делать, если это сочетание ни разу не нашлось? Очень просто: значит, сегодня отгрузки не будет.

Между прочим, программирование очень часто применяется для шифровки, и соответственно для взлома шифровок.

И самое феноменальное достижение математиков в области шифрования – это **криптография с открытым ключом**, которая позволяет собеседникам сначала сообщить друг другу открытые, несекретные ключи, а после этого преспокойно обмениваться зашифрованными сообщениями, которые почти невозможно взломать, причём перехват несекретных ключей ничего не даст злоумышленникам…

Кто заинтересовался этим чудом современной математики – милости просим в Интернет читать про алгоритм Диффи – Хеллмана, изобретённый ещё в 1976 году…

Как то ни странно, удалять элементы из списка можно не только методом `.pop()`, но и методом `.remove()`, вот только делают они это совершенно по-разному.

Если `.pop(i)` удаляет из списка **i-ый элемент**, то `.remove(X)` удалит элемент со **значением X**:

```
>>>Dig1=[1, 3, 4, 2, 77, 6, 2, 6]
>>>Dig1.remove(77)
>>>Dig1
[1, 3, 4, 2, 6, 2, 6]
```

Вот только если элемента нет в списке, то программа остановится по ошибке:

```
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
Dig1.remove(7)
ValueError: list.remove(x): x not in list
```

Поэтому этот метод лучше применять в связке с оператором **in**. Примерно вот так:

```
TT=[1,100.0,78,'ура', 12.23]
while True:
    x=float(input('что удалить: '))
    if x in TT: #если число есть в списке
        TT.remove(x)#удаляем число
        print ('число',x,'удалено')
        break #выходим из цикла
    else: #если числа нет - продолжаем
        print('такого числа нет в списке')
```

Запускаем программу, и беседуем с ней:

```
что удалить: 1.1
такого числа нет в списке
что удалить: 1
число 1.0 удалено
```

Упражнение 14.5

Список A длиной 10 элементов задаётся с помощью генератора случайных чисел. Заданный список надо сдвинуть на M элементов вправо по кругу, то есть последние элементы теперь будут первыми.

Упражнение 14.6

А давайте сосчитаем, какова вероятность совпадения дней рождения у двух людей в группе из 25 человек?

А теперь у вас есть знания, которые позволят вам создать давно ожидаемую программу раздачи карт для игры в "дурака".

Упражнение 14.7

Напишите программу, которая перетасует колоду в 36 карт, раздаст трём игрокам по 6 карт... Почему только трём? А четвёртый что-то сегодня не пришёл. Говорит, к экзамену по Питону надо готовиться.

Далее Питон распечатывает карты, находящиеся на руках у всех 3 игроков, показывает карту, определяющую козыри, и человеческим голосом ... извините, русским языком печатает название козырной масти и напоследок распечатывает, что осталось в колоде.

Помните, в начале книги я обещал вам рассчитать, когда следует отмечать миллионный День рождения – то есть, когда вы проживёте 1 000 000 дней? Вот давайте и выполним

Упражнение 14.8

Программа сначала запрашивает день, месяц и год рождения, затем – какой по счету ДЕНЬ рождения желаете рассчитать (не день в году, а число дней, прошедших со дня рождения!). Ответ надо вывести на экран в виде:

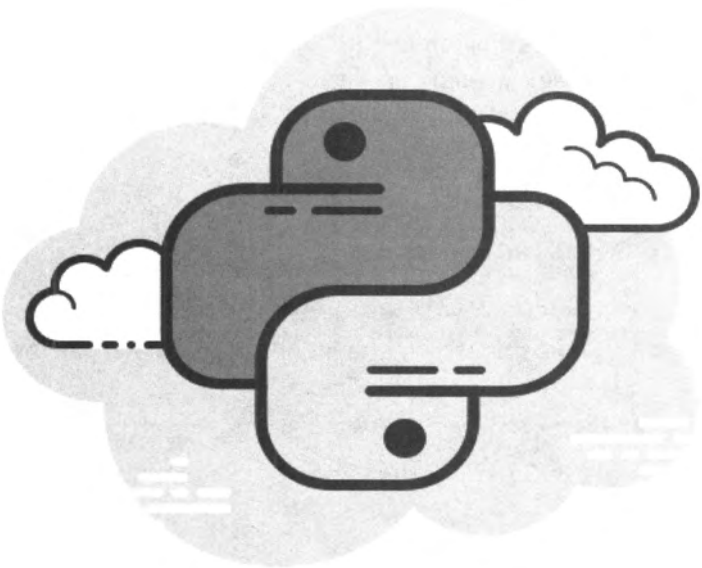
Ваш 1000000 день рождения отмечайте 2 октября 2049 года

Список – это последовательность, состоящая почти из всего, что угодно: чисел, текстов, списков, строк, кортежей, денег, кирпичей… ой, последние два, пожалуй, лишние… А самое главное – любой элемент в любой момент можно *переименовать*, не переименовывая весь список. А вот методов, обрабатывающих список, немного: всего-то 11 штук, и почти все они перечислены в **Приложении 2**.

У списков имеется, так сказать, двоюродный брат, и звать его **"Кортеж"**, с которым мы вот-вот познакомимся. Не переключайтесь!

Глава 15.

КОРТЕЖИ



Вы будете смеяться, но ***кортеж*** *– это тот же список, но доступный только для чтения.*

Я согласен с вами – создателей Питона иногда трудно понять. Но... "если звезды зажигают – значит – это кому-нибудь нужно?". Да, Владимир Владимирович[1], нужно!

Преимущества кортежей для непрофессиональных программистов практически незаметны, но всё же:

- Кортежи теоретически работают быстрее (то есть обращение к элементу кортежа занимает меньше времени). Но! Автор старательно сравнил скорость работы кортежей и списков... "Дурят нашего брата"[2]!.. Кортеж далеко не всегда работает быстрее списка...;
- Зато кортеж нельзя изменить по неосторожности;
- Кортежи требуют меньше памяти.

Создать кортеж очень просто – перечислением его элементов в круглых скобках:

```
>>>A1=(1,2,3)
>>>A1
(1, 2, 3)
```

1 Ну конечно же это Владимир Маяковский

2 Вы думали, что автор этой фразы – великий А. Райкин? Правильно, он её первый озвучил; а вот придумал эту фразу великий М. Жванецкий

А можно и через функцию **tuple()** (это и есть "кортеж" по-английски); она преобразует заданную *последовательность* в кортеж:

```
>>> A2=tuple("Кортеж")
>>>A2
('К', 'о', 'р', 'т', 'е', 'ж')
>>>A3=tuple([1,2,3])
>>>A3
(1, 2, 3)
```

А вот такой фокус не проходит:

```
>>>A4=tuple(1,2,3)
Traceback (most recent call last):
File "<pyshell#13>", line 1, in <module>
  A4=tuple(1,2,3)
TypeError: tuple expected at most 1 argument, got 3
```

Питон обиделся и говорит, что у функции **tuple()** должен быть 1 аргумент, а вы даёте аж 3! Всё верно – **tuple()** разбивает данную ему последовательность (Одну! И только одну!) на элементы и собирает из этих элементов кортеж.

Кортеж, как и список, поддерживает *операцию среза*, *конкатенацию* **(+)** и *проверку на вхождение* **(in)**.

Пример:

```
>>>T=1,'Щ',3,"t",5
```

Да! Кортеж можно задать и вот так – без скобок, перечислением содержимого через запятую.

```
>>>T[1]      #доступ по индексу
'Щ'
```

Обратный порядок

```
>>>T[::-1]
(5, 't', 3, 'Щ', 1)
```

При этом сам кортеж совершенно не меняется – он просто считывается задом наперёд!

- Срез:

```
>>>T[1:4]
('Щ', 3, 't') # кортеж не изменился
```

- Проверка вхождения:

```
>>>3 in T
True
```

- Конкатенация:

```
>>>T+(1,2,3)
(1, 'Щ', 3, 't', 5, 1, 2, 3)
```

Но сам кортеж **T** при этом упорно не желает меняться:

```
>>>T
(1,'Щ',3,"t",5)
```

А если всё-таки очень хочется изменить кортеж? Ну вот прямо очень-очень! Хорошо. Давайте сконкатенируем (во словечко-то у меня получилось!) 2 строчки в одну:

```
>>>BLOK=tuple('И снова бой!')
>>>Blok=tuple(' Покой нам только снится'[3])
```

3 Вспомнили, откуда цитата? Верно, из стихотворения А. А. Блока "На поле Куликовом" (1909)

```
>>>BLOK=BLOK+Blok
>>>BLOK
('И', ' ', 'с', 'н', 'о', 'в', 'а', ' ', 'б', 'о', 'й', '!', ' ', 'П', 'о', 'к', 'о', 'й', ' ', 'н', 'а', 'м', ' ', 'т', 'о', 'л', 'ь', 'к', 'о', ' ', 'с', 'н', 'и', 'т', 'с', 'я')
```

Как видите, если нельзя, но очень хочется – то можно. Просто надо изменённый кортеж обозвать каким-нибудь другим именем, а можно и тем же самым, как я проделал в примере выше. Нельзя менять отдельные элементы кортежа, а весь кортеж целиком – запросто.

Кортеж поддерживает всего 2 метода.

```
>>>BLOK.index('o')
4
```

Метод **.index()** возвращает номер первого вхождения в кортеж элемента с указанным значением. А вот метод **.count()** считает, сколько элементов с указанным значением входит в кортеж:

```
>>>BLOK.count('o')
6
```

Ого! Буква 'o' встречается аж 6 раз в этом небольшом отрывке! А как бы узнать, на каких местах? Очень просто –надо записать метод **.index()** в более полной форме:

```
>>>BLOK.index('o',5)
9
```

То есть через запятую мы сообщаем, с какого номера надо начинать поиск. А через следующую запятую мы сообщаем, до какого номера вести поиск (по умолчанию поиск ведётся да последнего посетите… извините, элемента):

```
>>>BLOK.index('o',10,15)
13
```

Хватит теории – переходим к практическим занятиям:

Упражнение 15.1

Пусть кортеж представляет собой вот такой стих:

Никогда, никогда ни о чем не жалейте —
Ни потерянных дней, ни сгоревшей любви.
Пусть другой гениально играет на флейте,
Но ещё гениальнее слушали вы.[4]

Напечатайте одной строкой номера позиций, где стоят буквы "д".

В главе 11 я вскользь[5] упоминал, что функция умеет возвращать несколько значений в форме кортежа. Вот вам на основе этого

Упражнение 15.2

Дан список чисел. Надо для каждого числа из этого списка распечатать его произведение на 2, квадрат числа и квадратный корень. Результаты следует рассчитать посредством функции.

Частенько кортежи состоят из чисел. Вот и давайте сделаем на вид очень простое[6] упражнение:

Упражнение 15.3

Дан очень простой, очень короткий (как вы понимаете, для Питона все последовательности с числом членов меньше 1 000 000 – короткие) кортеж:

4 Автор этого – согласитесь, потрясающего стихотворения – Андрей Дементьев

5 В этом слове, заметьте, на 8 букв приходится всего 1 гласная! Это рекорд для русских слов, если брать только приличные

6 В математике часто бывает, что простейшая на вид задача оказывается невероятно сложной. Как вам, например, проблема Гольдбаха (немецкий и российский математик XVII века): любое чётное число, большее 4, можно представить в виде суммы 2 простых чисел (6=3+3, 12=7+5, 100=97+3). Проблема ни доказана, ни опровергнута до сих пор!

a = (55, 31, 33, -40, 62, -27, 58, -2, 5, 9, -5, 33, 10)

В этом кортеже надо найти первые попавшиеся три числа, которые в сумме дают число 13. Далее составить из них новый, с иголочки, кортеж, напечатать его и на этом завершить выполнение программы. Если подходящая тройка не найдена – сообщите об этом.

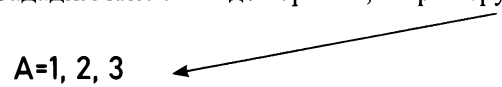
Чуть не забыл, но вы, наверное, уже и сами догадались, что число элементов кортежа можно узнать вездесущей функцией len().

Упражнение 15.4

Кортеж очень удобно использовать как ключ при шифровке донесений – здесь его неизменность очень важна! Вот и создайте программу, которая зашифрует донесение:

"Господствует ещё смешенье языков: Французского с нижегородским?[7]"

Хороший вопрос – каким образом будем шифровать? Давайте вместо символа из донесения использовать другой, отстоящий от него на определённое, заданное ключом, число шагов. Зададим ключ в виде кортежа, к примеру:



Это значит, что вместо первой буквы донесения подставим символ, отстоящий от неё на 1 шаг, вместо 2-ой – отстоящий от неё на 2 шага, вместо 3-го – на 3 шага, а вот вместо 4-го – вновь отстоящий на 1 шаг. Поэтому сообщение "АЛИСА" превратится в "БНЛТВ".

Ключ может быть любой длины; ещё надо создать кортеж со списком всех букв, цифр и даже символов русского алфавита, например:

7 Из комедия А.С. Грибоедова "Горе от ума", которая разошлась на огромное количество цитат

RUS=tuple("АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЪЭЮЯ
0123456789,.;:()/%!?абвгдеёжзийклмнопрстуфхцчшщьыъэюя")

Вот с помощью кортежа RUS мы и будем заменять символы, то есть если нам надо заменить букву "Я", а ключ в данный момент равен 1, то вместо "Я" ставим пробел, а если ключ равен 2, то вместо "Я" уже поставим 0…

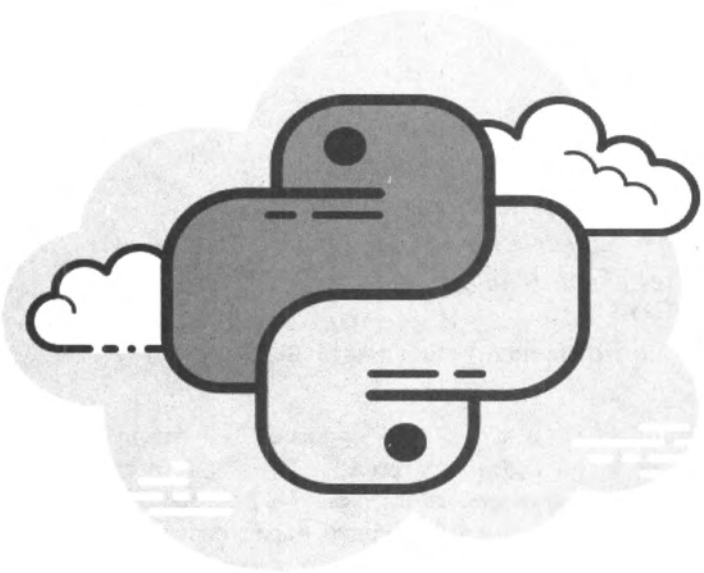
Кортеж – это предмет солидный, основательный – не то, что его легкомысленный и ветреный братишка *список*. Никаких изменений в себе он не терпит, и поэтому у кортежа только 2 метода: **.count** и **.index** – а как они работают – смотри в **Приложении 2**.

Если уж очень надо изменить кортеж, извольте сохранить результат под другим именем (а можно и под тем же. Мы, кортежи, существа покладистые).

Ну а теперь можно заняться **словарями**. Словарями? А какой язык будем учить? Никакой, хотя вы почти правы – словарь в Питоне иногда похож на традиционный…

Глава 16.

СЛОВАРИ



Допустим, нам надо создать англо-русский словарь. В рамках наших знаний сначала создаём список на английском:

```
DicEng = [ "one", "two","three"]
```

Затем список на русском:

```
DicRus = [ "один",  "два", "три"]
```

Ну а дальше:

Сначала в English ты зайдёшь,
Там нужный индекс ты найдёшь
Потом уж в русский перейдёшь
Потом значенье извлечёшь
- и делай с ним что хошь![1]

Ну а на Питоне это выглядит ещё прекрасней:

```
DicEng = [ 'one', 'two','three']# английские слова
DicRus = [ "один",  "два", "три"]# русские аналоги
wrd=input("переведи на русский: ")# вводим слово для перевода
IND=DicEng.index(wrd)# находим адрес слова в английском списке
Rus=DicRus[IND] # находим перевод
print (Rus) # печатаем перевод
```

1 Здесь автор позволил себе немного переделать песенку лисы Алисы и кота Базилио из фильма "Приключения Буратино". Слова Б. Окуджавы

К счастью, создатели Питона по доброте душевной избавили нас от этих сложностей, придумав такое понятие, как **СЛОВАРЬ.** Берём, к примеру, русско-английский словарь, открываем, находим слово "словарь", читаем перевод – "dictionary". Похожим образом используются словари и в Питоне!

По одному слову (называется "ключ") находится другое (называется "значение"), причём под словом понимается любой набор букв и цифр, а также и числа.

Объясняю на пальцах: во всех нормальных последовательностях (списки, строки и кортежи) его элемент вызывался по номеру (правда, нумерация начинается с нуля, но это уже пустяки, дело привычное).

В словаре же элемент вызывается, так сказать, по его названию (в Питоне это именуется "ключ"), прямо как в обычном словаре: "five"? – "пять"!

Пример – создадим англо-русский словарь:

```
>>>Dict = { "one" : "один", "two" : "два", "three" :"три" }
```

Доступ к значениям словаря осуществляется по ключу:

```
>>>Dict ["one"]
'один'
```

И напишем программу, аналогичную вышеприведённой:

```
Dict = { "one" : "один", "two" : "два", "three" :"три" }
wrd=input("переведи на русский: ")
print (Dict[wrd])
```

Коротко и понятно! На экране мы получим небольшой диалог:

```
переведи на русский: one
один
```

В словарь легко добавить запись:

```
>>>Dict["four"]="четыре"
>>>Dict
{'one': 'один', 'two': 'два', 'three': 'три', 'four': 'четыре'}
```

Но если ключ уже есть в словаре, ему будет присвоено новое значение:

```
>>>Dict["four"]="пять"
>>>Dict
{'one': 'один', 'two': 'два', 'three': 'три', 'four': 'пять'}
```

А вот удалять ключ из словаря надо специальным оператором **del** ("delete" – "удалять"):

```
>>>del Dict ["four"]
>>>Dict
{'one': 'один', 'two': 'два', 'three': 'три'}
```

При обращении по несуществующему ключу Питон вас вежливо отругает:

```
>>>Dict['five']
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    Dict['five']
KeyError: 'five'
```

(Приговаривая, наверное, про себя: "Трудно ловить чёрную кошку в тёмной комнате. Особенно, если там её нет"[2])

Так что желательно сначала убедиться в наличии ключа с помощью оператора **in**:

```
>>> "five" in Dict
False
```

2 Изречение приписывают китайскому философу Конфуцию, вот только в его трудах этой фразы почему-то не нашлось

Упражнение 16.1

Пусть у нас есть список сотрудников фирмы "Художники". У каждого художника свой табельный номер. Эти номера, мало того, что идут не подряд, так ещё и некоторые отсутствуют (ну вдруг человек уволился – тогда его табельный номер вычёркивается и другому не присваивается). Задача: найти в этом списке какого-нибудь Айвазо… И правда – пора найти Айвазовского и вывести на печать его табельный номер.

Осложнение: бестолковый пользователь может ввести фамилию с маленькой буквы – тем не менее, сотрудника в любом случае надо найти (в списке, а не в комнате, где он сидит).

Усложнение: если человека нет в списке, надо его туда ввести и присвоить новый табельный номер.

Дополнение: вводить нового человека надо с разрешения пользователя – вдруг человек (тот, который пользователь) ошибся в написании его (нового человека) фамилии. Подсказка: надо использовать методы *.isdigit()* и *.strip()*.

Замечание: разумеется, надо создать словарь, в котором каждому табельному номеру отвечает фамилия художника.

Окончание: программа работает постоянно (то есть организуем бесконечный цикл), пока у неё (программы?! Да нет же, у сотрудницы отдела кадров!) не закончится рабочий день, следовательно, программа завершает работу при вводе любой цифры (выходим из цикла оператором *break*).

Самый простой способ создать словарь – просто перечислить пары "ключ" : "значение" – но не самый удобный.

Но можно сделать это более элегантным способом – через функцию `zip()` ("застежка").

Например:

```
keys= ("apple", "cucumber", "melon", "raspberry", "pear")
Val=("яблоко", "огурец", "дыня", "малина", "груша")
fruits=zip(keys, Val) # объединяем 2 списка в список кортежей (17.1)
dicF=dict(fruits) # создаём из списка кортежей словарь
print (dicF)
```

Разберём, что у нас получилось.

Результат работы функции **zip()**, к сожалению, это вещь в себе, называется **объект**, и делиться содержимым так просто не желает, попробуйте **fruits** распечатать:

```
>>>fruits
<zip object at 0x000001E33C5A0800>
```

Что там написано? По-моему, "сам дурак!", хотя я могу и ошибаться. А вот функция **dict()** может с этим объектом-субъектом договориться – после запуска программы (17.1) видим на экране:

```
{'apple': 'яблоко', 'cucumber': 'огурец', 'melon': 'дыня',
'raspberry': 'малина', 'pear': 'груша'}
```

Я надеюсь, вы догадываетесь, что словари в Питоне используются не только как словари (прошу прощения за тавтологию)?

Ну например, давайте сосчитаем, сколько раз встречается каждая буква в классической фразе:

"...Бойцовый кот есть самостоятельная боевая единица сама в себе, способная справиться с любой мыслимой и немыслимой неожиданностью..."[3]

3 Великолепная железобетонная фраза-девиз из пророческого произведения Стругацких "Парень из преисподней"

С помощью словаря задача решается элегантно и просто:

```
TomCat = "Бойцовый кот есть самостоятельная боевая
единица сама в себе, способная справиться с любой мыслимой и
немыслимой неожиданностью"
ABC = {} # создаём пустой словарь
for i in range(1, len(TomCat) - 1): # проход по заданной фразе
    if TomCat[i] not in ABC:# если символа нет в словаре
        ABC[TomCat[i]] = 1 # добавляем символ в словарь как ключ
и задаём его значение как 1
    else: # символ есть уже в словаре
        ABC[TomCat[i]] += 1 # увеличиваем его значение на 1
print (ABC) # печатаем распределение частот
```

Запускаем и любуемся на результат:

```
{'о': 13, 'й': 5, 'ц': 2, 'в': 4, 'ы': 3, ' ': 20, 'к': 1, 'т': 6, 'е': 8, 'с': 13, 'ь': 4, 'а': 9, 'м': 6, 'я': 5,
'л': 4, 'н': 7, 'б': 4, 'д': 2, 'и': 7, ',': 1, 'п': 2, 'р': 1, 'ю': 1, 'ж': 1}
```

Упражнение 16.2

Пока любуетесь на результат, добавьте в программу пару строк, чтобы узнать, какие 2 символа встречаются чаще всего.

В своё время (вторая половина XX века) был очень популярен "Эрудит" – игра, где нужно составлять слова (русские, осмысленные) из букв, как в кроссворде, на специальной игровой доске, причём каждая буква имела свою стоимость (в баллах). Стоимость слова рассчитывалась как сумма стоимости букв, из которых оно состоит. Например, буква М давала 2 балла, буква А – только 1, поэтому слово МАМА давало игроку 6 баллов.

Упражнение 16.3

Напишите программу, которая посчитает для "Эрудита" стоимость слова, введённого с клавиатуры. Вот здесь-то словарь пригодится во всей красе!

Кроме того, нужна наша любимая "защита от дурака" – если введено слово из нерусских букв – проявляем патриотизм и пишем: "Слово должно состоять только из русских букв". Выход из программы пусть будет при нажатии любой цифры.

Словарь в Питоне – это вам помощнее обычного словаря будет! В нём вы можете содержать любые пары слов и чисел. И обращаться к значениям надо не по номеру, а по кличкам… извините, по ключам. И добавлять в него пары "ключ-значение" легко и просто. И методов в словаре всего-то 11. Основные, как обычно, в **Приложении 3**.

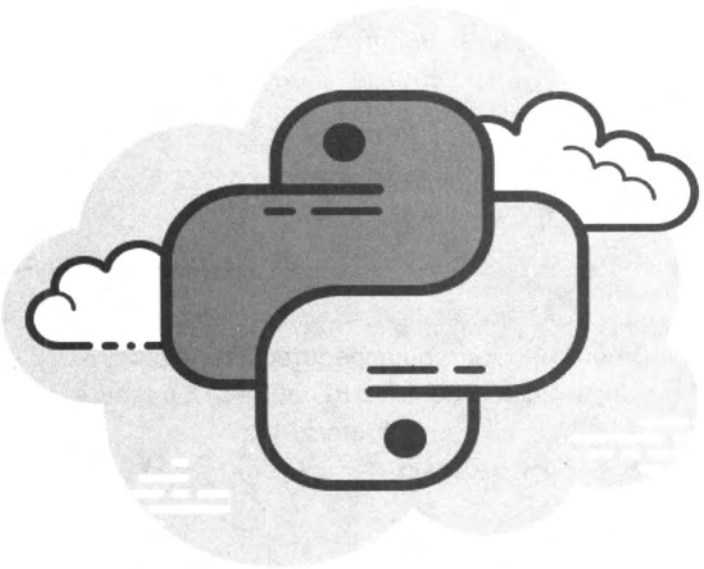
Ну вот, теперь вы знаете про Питон почти всё, что надо для начала. "И всё бы хорошо, да вот что-то нехорошо".[4]

Ну конечно же! Мы забыли о **множестве** – интереснейшем виде последовательности.

4 А это уже А. Гайдар, "Сказка о военной тайне, Мальчише-Кибальчише и его твёрдом слове"

Глава 17.

МНОЖЕСТВО



Множество – это набор уникальных элементов, да ещё и сложенных в беспорядке.

"Да это же свалка!" – скажете вы. Нет! Это называется "рабочий беспорядок" – вы же у себя на столе довольно быстро находите, что надо. А в Питоне в свал… – в множестве нужный элемент находится мгновенно!

Как обычно, начнём с примера. Возьмём великолепное (и заметьте, очень верное!) двустишие:

"Сомнений нет у идиота,
сомненья гложут мудреца"[1]

и поместим его в переменную **SOFO**:

```
>>>SOFO='Сомнений нет у идиота, сомненья гложут мудреца'
```

И сделаем из него множество посредством функции `set()` (это "множество" по-английски – на тот случай, если вы ещё не выучили английский):

```
>>>Sofo=set(SOFO)
```

1 Владимир Поляков – автор огромного количества остроумных (и просто умных) двустиший

Посредством любимой функции `len()` узнаем, сколько элементов в множестве:

```
>>>len(Sofo)
21
```

И посмотрим, что у нас получилось:

```
>>>Sofo
{' ', 'ц', 'и', 'ж', 'р', ',', 'с', 'т', 'ь', 'й', 'г', 'е', 'н', 'о', 'я', 'л', 'С', 'а', 'м', 'д', 'у'}
```

То есть получился просто список букв, использованных в подопытном афоризме! Да ещё и без всякого порядка! Ну и зачем городить новое понятие, которое даже хуже старого? А затем, что в множестве нужный элемент находится необыкновенно быстро!

Кто сказал "Не верю![2]"? Ах это вы, уважаемый Константин Сергеевич... Тут вам не МХАТ – сейчас докажу на примере!

Набираем на компьютере программу:

```
import random # грузим генератор случайных чисел
import time # грузим модуль времени
Rand=[] #Список, куда сложим случайные числа
for i in range(100000):#Создаём список из 100000 случайных чисел

    X=random.randint(1,20000) #случайное целое число из диапазона
1-20000
    Rand.append(X) # Добавляем это число в список Rand
SetRan=set(Rand) # Сделаем из списка множество и запомним его в
SetRan
T1=time.time() #засекаем время
for j in range(1,10000):
    j in Rand #ищем последовательные числа в списке
T2=time.time()-T1 #поиск в списке занял T2 секунд
print (T2) #печатаем время поиска в списке
```

2 Слова знаменитого режиссёра Станиславского К.С., создателя МХАТа, которые он произносил на репетиции, если игра актёров ему не нравилась. По системе Станиславского актер должен максимально вживаться в роль. Если в игре актёра Станиславский чувствовал ненатуральность, фальшь, то он восклицал "Не верю!"

```
T3=time.time() #снова засекаем время
for j in range(1,10000):
     j in SetRan  #ищем последовательные числа в множестве
T4=time.time()-T3 #поиск в множестве занял T4 секунд
print (T4) #печатаем время поиска в множестве
print ('множество обработано в',int(T2/T4),' раз быстрее')
```

Запускаем и читаем на экране:

```
время поиска в списке 1.6862566471099854
время поиска в множестве 0.0009968280792236328
```

и самое приятное (барабанная дробь!!!):

```
множество обработано в 1691 раз быстрее
```

Бурные аплодисменты! Занавес. Потрясённый Станиславский покидает нас, бормоча: "Какие страсти кипят в информатике! Куда там моему театру!... Никогда бы не поверил!..."

При запуске у вас на компьютере результаты будут несколько иные, время расчёта зависит от множества таинственных процессов, постоянно происходящих у нас в компьютерах.

Что мы делали в этой программе? Сначала записываем в *список* **Rand** 100 000 случайных чисел из интервала от 1 до 20 000, на его основе создаём *множество* **SetRan;** затем запускаем цикл проверки наличия чисел из интервала от 1 до 9999 сначала из списка, а затем из множества и смотрим, за какое время наш любимый Питончик проделает эти операции.

Я думаю (безнадёжный я оптимист!), что все поняли – модуль **time** как-то занимается измерением времени – и тут вы совершенно правы!

Метод **time.time()** сообщает, сколько секунд прошло с начала эпохи.

Да нет, не с Рождества Христова – тогда в качестве компьютера использовалась счётная доска – абак. И самые лучшие компью… э… абаки стояли во дворцах царей и выглядели как доска из слоновой кости с сапфировыми и

рубиновыми камнями. Производительность этих первых счётных устройств, конечно, была маловата…

Так вот, у программистов эпоха начинается с 1 января 1970 года (я вас предупреждал, что мышление у программистов немножко сильно отличается от мышления нормальных людей).

Метод `time.time()` возвращает нам, сколько секунд прошло с начала эпохи в момент обращения к нему, причём с невероятной точностью.

Итак, ***поиск в множестве происходит очень быстро***; а чем ещё они хороши?

Давайте для примера возьмём теперь одностишье :

Тебя сейчас послать или по факсу?[3]

Поместим его в переменную **VISH**:

```
>>>VISH="Тебя сейчас послать или по факсу?"
```

И сделаем из неё множество:

```
>>>Vish=set(VISH)
```

Вы не забыли, что у нас есть ещё множество **Sofo**? Давайте узнаем, какие буквы есть как в одностишии, так и в двустишии:

```
>>>Vish&Sofo
```

Пожалуйста :

```
{'с', 'у', 'л', 'й', 'о', 'т', 'е', ' ', 'ь', 'я', 'и', 'а'}
```

3 Автор - Владимир Вишневский. Он и создал этот оригинальный жанр – одностишья

То есть оператор **&** (амперсанд) – это пересечение множеств. А вот объединению множеств достался довольно странный знак | (вертикальная черта):

```
>>>Vish|Sofo
{'с', 'й', 'п', 'р', 'у', ',', 'ф', 'д', 'и', '?', 'м', 'Т', 'к', 'л', 'о', 'ц', 'т', 'е', ' ', 'г', 'ж', 'б', 'ь', 'я', 'С', 'н', 'а', 'ч'}
```

Почему создатели Питона постеснялись использовать для данной операции обычный знак "+" (плюс) – непонятно. "Есть многое на свете, друг Горацио, что и не снилось нашим мудрецам"[4]. А вот знак "-" (минус) – пожалуйста, работает:

```
>>>VS=Vish-Sofo
{'п', 'б', '?', 'Т', 'ч', 'ф', 'к'}
```

Эта операция удалила из множества **Vish** все элементы, которые есть и во множестве **Sofo,** и загрузила их в множество **VS.**

Ну и наконец оператор ^ (стрелочка вверх) выдаст нам элементы обоих множеств, исключая *одинаковые:*

```
>>>Vish^Sofo
{',', 'н', 'ж', 'ц', 'к', '?', 'г', 'м', 'С', 'п', 'ф', 'д', 'р', 'Т', 'ч', 'б'}
```

У множества всего-то 5 методов:

```
>>>VS.add(100)
{'п', 'б', '?', 100,'Т', 'ч', 'ф', 'к'}
```

Добавили число 100 в множество, причём Питон помещает новый элемент куда ему заблагорассудится. А вот для удаления элементов придумано аж 3 (!) метода (а ещё говорят "ломать – не строить!". Создатели Питона, видимо, придумали метод для удаления, потом забыли про это, и создали ещё пару методов).

4 Шекспир, «Гамлет». Перевод Н. Полевого. Почему-то в качестве фразеологизма используется именно этот вариант перевода, несмотря на то, что существует более 20 других переводов

Методы `.remove(x)` и `.discard(x)` удаляют элемент `x` из множества; а отличие между ними почти незаметное: если удаляемого элемента нет в множестве, то в первом случае произойдёт ошибка и программа остановится, а во втором случае ничего не произойдёт и программа преспокойно продолжит работу:

```
>>>VS.remove('к')
>>>VS
{'п', 'б', '?', 100,'Т', 'ч', 'ф'}
>>>VS.discard('?')
{'п', 'б', 100,'Т', 'ч', 'ф'}
>>>VS.discard('к')
>>>VS
{'п', 'б', 100,'Т', 'ч', 'ф'}
```

А вот метод `.pop()` удаляет произвольный элемент (то есть какой Питону захотелось) и возвращает его:

```
>>>VS.pop()
'ф'
>>>VS
{'п', 'б', 100,'Т', 'ч'}
```

Ну и последний метод просто очищает множество:

```
>>>VS.clear()
>>>VS
set()
```

Так как множество – неупорядоченный набор элементов, то обращаться к его элементам по номерам не получится:

```
>>>Vish[5]
Traceback (most recent call last):
```

```
  File "<pyshell#4>", line 1, in <module>
Vish[2]
TypeError: 'set' object is not subscriptable
```

Перевожу смысл последней строки: "Ошибка типа: объект "множество" не нумеруется".

Но, тем не менее, содержимое множества можно последовательно перебрать (пишем простенькую программку):

```
Gogol="И какой же русский не любит быстрой езды?"[5]
GGL=set(Gogol)
for i in GGL:
    print (i, end=' ')
```

запускаем и Питон послушно выдаёт всё содержимое множества GGL, причём в своём тайном порядке:

```
е д ю б о  й т и к з ы с И н ? а ж р л у
```

Между прочим, в множествах прекрасно работают функции **max, min** и, разумеется, **len**:

```
>>>len(GGL)
21
>>>max(GGL)
'ю'
>>>min(GGL)
' '
```

Понятно, что с буквами операторы **max** и **min** работают как-то туманно. А вот с числами дело пойдёт веселей:

```
>>>Numb={2,2.21567,7.11,-0.213}
```

Да, множество можно создать напрямую, записав его элементы в фигурных скобках.

5 Ну это, конечно же, Н. В. Гоголь, и конечно же, "Мёртвые души"

А вот со строками этот фокус не проходит:

```
>>>RURU={'Буквы'}
>>>RURU
{'Буквы'}
```

То есть для разбиения слова на буквы надо применить оператор **set** непосредственно к строке:

```
>>>set('Буквы')
{'у', 'в', 'ы', 'к', 'Б'}
```

Вернёмся к переменной **Numb**, а то она заскучала:

```
>>>max(Numb)
7.11
>>>min(Numb)
-0.213
```

Даже функция **sum** здесь работает:

```
>>>Sum (Numb)
11.112670000000001
```

Как я писал в упражнении 15.2, Питон порой ошибается в 16-ом знаке после запятой.

Я вижу, вы уже перегружены знаниями о множествах – значит, надо отдохнуть, сделать упражнение.

Упражнение 17.1

Определить, являются ли слова:

отсечка
сеточка
тесачок
чесотка

анаграммами, то есть состоят из одних и тех же букв.

Вообще-то говоря, применение множеств для определения, является ли одно слово анаграммой другого, не совсем правильно. В анаграммах иные буквы могут повторяться, например:

лепесток - телескоп

А вот для панграммы (это короткий текст, использующий все буквы алфавита, по возможности не повторяя их) понятие множества как нельзя подходит. Самый известный пример панграммы:

Съешь же ещё этих мягких французских булок да выпей чаю.

Упражнение 17.2

Напишите программу, которая проверяет, является ли заданный текст панграммой. На экран следует вывести текст: "Это панграмма" и количество букв в ней, либо просто "это не панграмма"

Упражнение 17.3

Вооружённые новыми знаниями, создайте программу "Морской бой". Нет, не настоящий "морской бой" на 2

бумажках, а будете искать, куда Питон расставил свои корабли. Значит, так: программа сначала случайным образом выдаёт координаты (буква + цифра) и запоминает их в множестве, а затем предлагает вам пострелять. Если выданные вами координаты есть в множестве – значит попал. Нету – промазал. При нажатии какой-нибудь редкой буквы игра заканчивается и Питон печатает результаты: число попаданий и промахов. Да-да, вы правы: получается, что все корабли у Питона – однопалубные.

Упражнение 17.4

Для игры в "Морской бой", конечно, использование множества некритично – слишком небольшой размер поля. А вот теперь представьте ситуацию: у вас есть автоматизированный склад на 10 000 000 ячеек, и вам надо как можно быстрее найти пустую ячейку поближе к выходу (это чтобы снизить пробег погрузчика – склад-то огромный, а электроэнергия для зарядки погрузчика не бесплатна), чтобы отправить туда робот-погрузчик-штабелёр с поступившим грузом. Вот тут понятие множества и пригодится – работает быстро и содержит только уникальные (номер ячейки уникален) элементы. Поэтому напишите эту очень нужную в экономике программу.

Входные данные таковы: на складе 10 000 000 ячеек; нумерация ячеек вида x-y-z, где x и y меняются от 1 до 1000, а z от 1 до 10. Если ячейка занята, то её номер есть в множестве SKLAD, свободна – номера нет. Сначала датчиком случайных чисел заполните склад на 99%, а затем сообщите № свободной ячейки поближе к выходу (чем меньше координата x или y – тем ближе к выходу находится ячейка).

А вот **множества** мне очень нравятся – за феноменальную быстроту работы и уникальность значений. А методов всего-то 5, я лучше их просто напомню:

add – добавляет;

remove, discard, pop – удаляют;

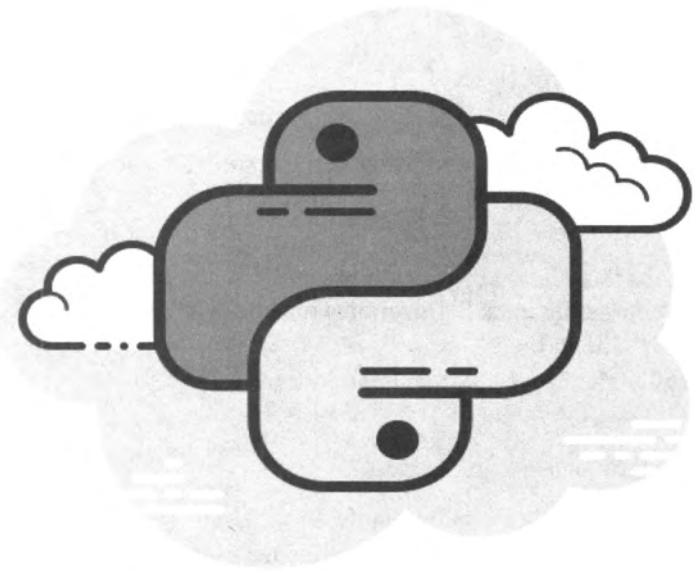
clear – очищает.

При написании программ вы заметили, что иногда Питон спотыкается об ваши же ошибки, ругается и прекращает работу, так что порой хочется ругнуться в ответ и сказать: "Что же ты, Питон, такой бестолковый? Ошибка несущественная, пропусти её и считай дальше!!!".

Итак, что делать, если случилась авария? (раньше такие явления назывались "авост ЭВМ" – аварийная остановка ЭВМ. Сейчас этот термин вышел из употребления. Вместо него употребляется термин "А, опять подвисла, зараза…").

Глава 18.

ЧТО-ТО ПОШЛО НЕ ТАК



Допустим, вам жизненно необходимо поделить 16,202 на разность между 17,4 и числом, вводимым с клавиатуры. Вы быстренько (ведь недаром же вы почти всю эту книжку прочитали?) написали вот такую милую программу:

```
Ntt=16.202
U_u=17.4
t=float(input('Жду число '))
K2K=Ntt/(U_u-t)
print(round(K2K,2))
```

Программа хорошо, устойчиво работает. Вводим 11 – получаем 2.53, вводим 17 – на экране 40.51, вводим 17.4... – ба-бах!

```
Traceback (most recent call last):
File "C:\Как бы диск D\Питоновы программы\try.py", line 23, in <module>
K2K=Ntt/(U_u-t)
ZeroDivisionError: float division by zero
```

По-моему, Питон чем-то недоволен...

Ну точно – в последней строке написано: "НольДелениеОшибка: текущее деление на ноль". Всё правильно – на 0 делить ни один компьютер не умеет. А так всё было хорошо! Что делать? Проще всего не допускать, чтобы `t` принимало значение 17,4 – это легко сделать, благо знаменатель у нас

простой. А теперь представьте, что знаменатель имеет вид, к примеру, вот такой:

```
((U_u*i2i2+qR2*pek4a)-t)
```

и все переменные – переменные (прошу прощения за тавтологию). Таким образом, деление на ноль здесь вполне может случиться, а в какой момент – неизвестно, кроме того, что в самый неподходящий.

Вот для таких случаев, когда мы догадываемся, что неприятность может случится, но не знаем – когда, и предназначен двойной оператор **try / except** (в переводе с английского: "пытаться"/"исключение"). Операция, которая может вызвать исключение, помещается внутрь блока **try**. А блок, который должен быть выполнен при возникновении ошибки, находится внутри **except**.

Перепишем нашу программу:

```
Ntt=16.202
U_u=17.4
t=float(input('Жду число '))
try:
    K2K=Ntt/(U_u-t)
    print(round(K2K,2))
except ZeroDivisionError:
    print ('случилось деление на ноль')
print ('продолжим работу')
```

С помощью оператора **try** я "пытаюсь" выполнить деление. Если деление прошло нормально, то я вывожу результат на экран и продолжаю работу. А вот если случилось деление на 0, то программа перепрыгивает на оператор **except** ("исключение"), где и происходит обработка ошибки (просто печатаем **'случилось деление на ноль'** и продолжаем работу). Прерывания работы программы не произошло – и это главное!

Например, вводим 12, на экране:

```
3.0
продолжим работу
```

А теперь 17.4:

```
случилось деление на ноль
продолжим работу
```

Всё прекрасно, деление на 0 мы обошли!

Небольшой нюанс: вы обратили внимание, что после **except** стоит странное выражение **ZeroDivisionError**?

Когда программа при выполнении наталкивается на ошибку, то всегда указывает, в какой строке, какой программы и какого рода ошибка произошла:

```
Traceback (most recent call last):
File "C:\Как бы диск D\Питоновы программы\try.py", line 23, in <module>
K2K=Ntt/(U_u-t)
ZeroDivisionError: float division by zero
```

Я выделил название ошибки. Дело в том, что ошибки могут быть самые разные (примерно 40 видов), и обрабатывать их порой надо тоже по-разному. Вот, например, простая программа – вводим 2 целых числа, и делим одно на другое.

Здесь возможны **два вида ошибок**:

1. Вместо числа введена буква, а то и какой-нибудь знак.
2. Деление на ноль.

и обрабатывать их надо по-разному.

Как называется ошибка деления на ноль, мы знаем:

ZeroDivisionError.

А как узнать название *ошибки преобразования типа* – это когда введено не число, а буква?

Проще пареной репы! Даже программу писать не надо:

```
>>>D=input()
      два
>>>int(D)
```

В переменную **D** введено строковое значение **'два'**, при попытке преобразовать его в число мы получим ошибку:

```
Traceback (mostrecentcalllast):
File "<pyshell#1>", line 1, in<module>
  int(D)
ValueError: invalid literal for int() with base 10: 'два'
```

Вот вам и название ошибки: **ValueError**!

Тогда вот вам и программа с обработкой 2 видов ошибок:

```
while True:
      try:
            b = int(input('b = '))
            c = int(input('c = '))
            print ('результат деления: ',round (b/c,
      3))
            break
      except ZeroDivisionError:
            print('Деление на 0. Введите число, отличное от нуля')
      except ValueError:
            print('Введите, пожалуйста, число')
```

Между прочим, возможна инструкция **except** без аргументов, которая перехватывает вообще все возможные ошибки.

По-моему, нам пора размяться и применить полученные знания. Сделаем-ка простое

Упражнение 18.1

Допустим, у нас есть огромное множество, состоящее из самых разных элементов: чисел, строк... Нам позарез вдруг приспичило удалить из этого множества какой-нибудь элемент (его вводим с клавиатуры). О выполнении задания следует доложить кому следует (вывести сообщение на экран). А если элемента нет? Об этом безобразии тоже следует сообщить на экран.

Подсказка: для удаления следует применить метод *.remove(x)*.

Мы уже сталкивались (когда находили углы треугольника) с ситуацией, когда нам надо получить от пользователя (сегодня уже от бухгалтера, готовящего годовой отчёт) число – целое или с плавающей запятой и отбросить все другие возможные варианты ввода. Конечно, самый простой вариант вот такой:

```
N=float(input ('введите число:'))
print(N)
```

и всё работает прекрасно – преобразуем полученную строку в число с плавающей запятой и дальше его преспокойно обрабатываем. Но! А где защита от дурака? Малейшая ошибка пользователя (по ошибке нажал клавишу с буквой. Скажете, так не бывает? Ещё как бывает![1]), и на экране появится страшная надпись:

1 Если какая-нибудь неприятность может произойти, она случается (основополагающий закон Мерфи)

```
Traceback (mostrecentcalllast):
File "C:/Как бы диск D/Питоновы программы/try.py", line 2, in <module>
   N=float(input ('введите число:'))
ValueError: could not convert string to float: 'д'
```

и пользователь ищет программиста, написавшего эту программу, с целью ~~поблагодарить~~ убить его страшной смертью (тысяча чертей! завтра сдавать годовой отчёт!!!!!). Питон тихонько хихикает в сторонке – он расписался в своём неумении: «Не могу преобразовать строку в число: 'д'», а нам с вами надо как-то разрулить эту ситуацию.

Конечно, самое простое, написать примерно такую программу:

```
while True: #бесконечный цикл в ожидании ввода ЧИСЛА
try:
      N=float(input ('введите число: '))
      print(N) #печатаем введённое число
      break #и завершаем программу
except ValueError :
      print ('Введено не число')
```

Но это нас не спасает, от другой, совершенно дурацкой ситуации! А что если пользователь вместо разделителя "точка" ввёл "запятую":

```
введите число: 5,5
```

и потрясённая новенькая бухгалтерша, ещё не знакомая с причудами нашей программы, изумлённо читает на экране:

```
Введено не число
```

Она ведь ввела число! Что этому тупому компьютеру не нравится? Как всякий нормальный человек, она повторяет ввод:

```
введите число: 5,5
Введено не число
```

Дальше крики "Идиотская программа!", "Уволюсь!"... А кто виноват? Программист? Нет! Мне сказали: "напиши фрагмент программы, которая отсекает ввод пользователем некорректных данных, а корректные данные – это числа, целые или с плавающей запятой."

Для пишущего программы на Питоне – 5.5 – это число, а 5,5... Э... Тоже число, но *некорректное*!!! Программист наивно полагает, что всё человечество прекрасно знает, что то, что на русском языке называется "числом с плавающей запятой" на самом деле выглядит ... как число с точкой! У нас всё так делается – через ... точку.

Придётся продолжить поиск виноватого... так вот же он! Постановщик задач! Именно этот человек обсуждает с заказчиком задачу, учитывает все его капризы, а также выявляет все возможные и невозможные проблемы и затем объясняет задачу программисту: что он должен создать и почему за столь малый срок. Вот он-то (постановщик задач, конечно) и *обязан* был предусмотреть подобную ситуацию и разъяснить программисту, что 5,4 и 5.4 – это одно и тоже число, как это ни странно.

Конечно, самое простое решение – это переложить проблему на клиента:

```
N=float(input('введите число (десятичные числа вводите через точку):'))
```

Это не есть хорошо![2] Мы-то с вами, в отличие от Питона, понимаем, что хотел сказать пользователь – не зря же относимся к виду "Человек разумный".[3]

Поэтому придётся переписать программу примерно таким образом:

```
while True:
    N=input ('введите число:')# вводится строка!
    if ',' in N: # Проверяем наличие запятой вместо точки
        N=N.replace(',','.')#заменяем запятую на точку
    try:
        N=float(N)# преобразуем строку в число
        print(N)
        break #и покидаем программу
    except: #не смогли сделать из строки число
        print ('Это не число. Повторите ввод')
```

2 Ироничное подражание разговору иностранцев

3 А знаете ли вы, что в своё время, каких-то 100 000 лет назад, на Земле существовало как минимум 4 биологических вида человека: неандертальцы; человек флоресский - карликовый представитель рода Homo с острова Флорес, известен также как "хоббит"; денисовцы – жили в Денисовой пещере на Алтае; и наконец, Homo Sapiens, то есть мы с вами

Таким образом программа ожидает от клиента строки, состоящей из цифр, запятых и точек, превращает все введённые запятые в точки и пытается полученную строку превратить в число. Это всё равно может не получиться – но тут уж любой бухгалтер поймёт, что записи "112036,,11" и "Люблю Васю" никак не тянут на числа; и просьбу "**`Повторите ввод`**" воспримет спокойно.

Упражнение 18.2

Допустим, что у нас есть фрагмент огромной программы, в котором к строчному значению переменной abc добавляется значение переменной Dobo, которая иногда таинственным образом принимает численное значение, что вызывает появление ошибки и остановку программы. Программа огромная! Пишется многими людьми из разных стран! И кто в ней накосячил – разбираться некогда – завтра сдавать проект! Надо просто обойти эту проблему, а уж дальше найти виновного и ....!!!!![4]

Совершенно неожиданная возможность есть в Питоне – подложить соломку в те места, где ожидается прилёт тела программиста. Определяем опасные места в программе, и с помощью неразлучной парочки **`try / except`** благополучно избегаем остановки.

Пока что все наши программы – эдакая вещь в себе, а современные программы, как правило, обрабатывают файлы, пришедшие из других источников. Посмотрим, как это делается.

4 "Дальше следует непереводимая игра слов с использованием местных идиоматических выражений" – бессмертная фраза из "Бриллиантовой руки"

Глава 19.

ИЗВНЕ

ИЗВНЕ[1]

Для открытия (то есть для загрузки в программу) файла в Питоне используется функция **open()** ("открыть" в переводе с английского):

```
F1f=open('test.txt', mode)
```

Здесь **test.txt**– это название открываемого файла, а **mode** – подсказывает Питону, в каком режиме мы будем дальше работать с файлом (чтения и/или записи. Если ничего не указано, файл открывается в режиме чтения). Ну, а **F1f** – это переменная, где теперь лежит содержимое файла?

Как бы не так! **F1f** сейчас – это файловый объект! К нему нельзя подступиться также запросто, как к переменной какой-нибудь.

Это – конструкция гордая, даже распечатываться не желает; наберите:

```
>>>print (F1f)
```

и полу́чите набор местных идиоматических выражений языка Питон:

```
<_io.TextIOWrapper name='test.txt' mode='r' encoding='cp1251'>
```

(эти питонистые ругательства я даже переводить не берусь).

1 Это отсылка к названию первой повести А. и Б. Стругацких – о посещении Земли Пришельцами…

Как из **F1f** сделать нормальную переменную я расскажу немного позже, а пока давайте создадим файл **test.txt**. Напишите в блокноте, что-нибудь умное, например "Hello, world!". Зачем вы взяли бумажный блокнот? Питоны не едят бумагу!

Сделайте запись в компьютерном Блокноте (или любом текстовом редакторе) и сохраните *в той же папке*, где лежат все ваши программы на Питоне под именем test.txt, естественно. В окне, когда записываете название файла, посередине есть запись "Кодировка". Убедитесь, что там написано UTF-8. Сохранили? Пишем программку:

```
F1f=open('test.txt')
```

А вот теперь давайте прочитаем содержимое файлового объекта **F1f** методом **.read()**, а полученный результат присвоим уже нормальной переменной **TXT**:

```
TXT=F1f.read()
print (TXT)
F1f.close()
```

Да, после того как вы поработали с файлом, его необходимо **закрыть** ("close" – в переводе с английского "закрыть").

Почему? Освобождается память компьютера. Ладно, у нас пока что файл малюсенький, а потом мы будем работать с файлами в миллионы знаков! Так что привыкайте: открыли – закрыли!

Запускаем программу… Красота:

```
Hello, world!
```

Автор, чрезвычайно довольный, хихикает в уголке на пару с Питоном: во всех учебниках "Hello, world" идёт в самом начале, а у меня появляется только в конце книги !...

А почему фраза на английском? Давайте добавим что-нибудь по-русски, например, напишем в файле **test.txt** на следующей строке "Я знаю Питон!".

Запускаем программу, и ...

```
Hello, world!
РЇ Р·PSP°СЋ РџРёС‚PsPS!
```

А вот не надо было хвастаться! Не знаем мы ещё Питон!..

Что же произошло?

Как вам известно из 14 главы, каждая буква в компьютере записывается в двоичном коде, например, ***q*** – это последовательность из 8 бит:

00101011

Первоначально 8 единиц и ноликов хватало, чтобы закодировать все английские буквы (прописные и строчные), цифры и всякие прочие знаки препинания и плюсы-минусы, но потом компьютерщики обнаружили, что есть ещё иврит, арабский и прочие экзотические языки, на кодировку символа отвели уже 16 бит (а то и больше) – и вот тут-то и возникли разные способы представления букв русского алфавита. В результате **Блокнот** сохраняет символы в одной кодировке, а Питон использует другую.

Подскажем Питону, в какой кодировке хранятся наши файлы:

```
F=open('test.txt', encoding = "utf-8")
```

Далее как обычно:

```
TXT=F.read()
print (TXT)
```

и на экране – ура! – то, что надо:

```
Hello, world!
Я знаю Питон!
```

Не всегда нам надо читать весь файл целиком, можно считать только несколько символов (но почему-то только с начала):

```
F=open('test.txt', encoding="utf-8" )
TXT=F.read(5)
print (TXT)
```

На экране:

```
Hello
```

С другой стороны, если вы знаете, откуда начинается ваша информация, можно просто промотать лишнее:

```
F=open('test.txt', encoding="utf-8" )# открываем файл
TXT=F.read(21) # промотаем 21 символ
TXT=F.read(5)
print (TXT) # и печатаем остальные
```

на экране:

```
Питон
```

А вот метод **.readline()** ("прочитать строку" в переводе) считывает одну полную строку из указанного файла за раз, концом строки считается символ переноса '\n', либо конец файла. Этот символ переноса строки тоже будет присутствовать в строке, для считывания следующей строки метод надо применить ещё раз.

Вот, допустим, я создал в **Блокноте** файл с небольшим стихотворением под названием **Ananas.txt**. Считаем его построчно:

```
F=open('Ananas.txt', encoding="utf-8" )# открываем файл
TXT=F.readline() #считаем 1 строку
print (TXT) #печатаем
TXT=F.readline() #считаем 2 строку
print (TXT) #печатаем
```

```
TXT=F.readline() #считаем 3 строку
print (TXT) #печатаем
TXT=F.readline() #считаем 4 строку
print (TXT) #печатаем
F.close()
```

На экране:

На медведя вы, сосна,

Шишку бросили со сна,

А на нас, а на нас

Уроните ананас![2]

На экране строчки разделены пустой строкой. Это из-за того, что один перенос идёт из прочитанной строки (Блокнот в конце каждой строки обязательно добавляет знак переноса '\n'), а второй добавляется самой функцией **print**. Так что **print** удобнее записать так:

```
print (TXT,end="")
```

а программу переписать так:

```
def readly():
    TXT=F.readline() #считаем 1 строку
    print (TXT, end='') #печатаем
    return
F=open('Ananas.txt', encoding="utf-8" )# открываем файл
for i in range(4):
    readly()
F.close()
```

Печать каждой строки осуществляется пользовательской функцией без аргументов (так можно!), а на экране мы наконец-то видим то, что я записал в файл **Ananas.txt** в нормальном виде:

2 Смешные, каламбурные стихи Я. Козловского (каламбур – игра слов с использованием разных значений одного слова или разных слов или словосочетаний, сходных по звучанию)

На медведя вы, сосна,
Шишку бросили со сна,
А на нас, а на нас
Уроните ананас!

А если я хочу считать сразу весь файл построчно? Да нет проблем! Метод **.readlines()** ("прочитать строки") к нашим услугам!

```
F=open('Ananas.txt', encoding="utf-8" )# открываем файл
TXT=F.readlines() #считаем файл построчно
print (TXT)  #печатаем
F.close() # закрываем
```

Но на экране какой-то бардак:

['На медведя вы, сосна,\n', 'Шишку бросили со сна,\n', 'А на нас, а на нас\n', 'Уроните ананас!\n']

Объясняю в чём дело: метод **.readline()** считывает файл построчно в *строчную* переменную, а **.readlines()** – в *список*!

Вспоминайте, чем эти типы данных отличаются, так что вот вам и упражнение:

Упражнение 19.1

Переделайте последнее упражнение так, чтобы замечательный стих Я. Козловского был выведен на экран как положено – построчно. Обязательно используем метод *.readlines()*.

Файл мы открываем, конечно, не для того, чтобы его распечатать, нет! В файл после его открытия, как правило, вносятся какие-то изменения, после чего изменённый файл записывается под старым (или новым) именем

и закрывается. К примеру, возьмём вот такое четверостишие (не всё же нам одно- и двустишиями заниматься! Надо расти!):

В наш век искусственного меха
и нефтью пахнущей икры
нет ничего дороже смеха,
любви, печали и игры.[3]

И запишем его в файл garik.txt с помощью программы. Это очень просто:

```
text = '''В наш век искусственного меха
и нефтью пахнущей икры
нет ничего дороже смеха,
любви, печали и игры'''
garik = open('garik.txt', 'w')# откроем файл для записи
garik.write(text) # запишем в файл
garik.close()# закроем файл
```

Сначала в строку **text** запишем текст (слишком длинный текст, в котором есть переносы строки, записываем с помощью трёх апострофов в начале текста и трёх в конце). Затем откроем файл, в который желаем записать информацию, причем, если такого файла нет, он будет создан.

По умолчанию файл открывается для чтения, а если мы собираемся что-то в него записать, то надо указать режим чтения 'w' (write – писать); если надо в него что-то добавить – режим 'a' (add- добавить); читать и записывать – режим '+'.

Затем записываем то, что надо, в *файловый объект* (это ещё не в файл! Пока в некий промежуточный объект), а файл будет создан окончательно только после выполнения метода **.close()**. Находим в папке, где вы сохраняете все наши программы, файл **garik.txt**, открываем – вот он наш гарик!

Ну а теперь поработаем с другим текстом, совершенно изумительным:

3 Автор - Игорь Губерман. Он известен своими ироничными и очень точными четверостишиями, которые называют "гариками"

Чудная картина,
Как ты мне родна:
Белая равнина,
Полная лу2на,

Свет небес высоких,
И блестящий снег,
И саней далёких
Одинокий бег.[4]

Почему изумительным? Да потому, что в этом полном движения и жизни стихотворении нет ни *одного глагола*! Проверили? Убедились в волшебстве А. Фета? Так вот в это волшебное стихотворение агенты иностранной разведки (ну а кто же ещё?) зачем-то запихнули цифру, как видите, и сохранили в файле **fet.txt** (проделайте то же сами, уважаемые читатели).

Наша задача – прочитать файл, найти в нём цифру (причём мы не знаем, ни какая это цифра, ни где она находится), удалить её и перезаписать исправленный файл, после чего показать язык иностранным разведкам.

```
F=open('fet.txt', 'r',encoding="utf-8")# открываем файл для
чтения
TXT=F.read() #считаем файл построчно
F.close()# Закроем файл
for j in '0123456789': #ищем цифру в тексте
    if j in TXT: #цифра найдена
        break #прерываем цикл - мы нашли цифру!
txt=TXT.split(j) #разделяем строку на 2 по найденной цифре
F=open('fet.txt', 'w',encoding="utf-8")# открываем файл для
записи
T2=txt[0] + txt[1] #сливаем 2 строки в одну
F.write(T2) #записываем новую строку в файл
F.close() #закроем файл
```

Откройте файл **fet.txt** и убедитесь – мы сделали это!

"Ну и что же?" – скажет мне читатель
Удивлённый простотой задач.
"Я б руками всё быстрее сделал
Посложнее, автор, озадачь!"

4 Ну а здесь автор – Афанасий Фет

Вот так! Я даже на стихи перешёл, вдохновившись Губерманом и Фетом! Хорошо, вот вам задача, которую вручную решить невозможно:

Упражнение 19.2.1

Создайте строку длиной в 100 000 (Сто тысяч !) символов из букв и ,н, п, о, т, идущих в случайном порядке, и сохраните её в файл с каким хотите именем. Ага! Вручную как-то тяжело... Создавайте с помощью цикла и модуля `random`, предназначенного, если ещё не забыли, для генерации случайных чисел.

Упражнение 19.2.2

Откройте файл с сохранённой строкой в 100 000 символов и найдите, сколько раз в этом файле идущие подряд буквы образовали случайно слово "питон".

Ну а теперь допустим, что открываемого файла нет! Ну или совсем нет, или он лежит в другом каталоге... Тогда вы увидите на экране:

```
Traceback (most recent call last):
File "C:\Как бы диск D\Питоновы программы\питон.ру", line 12, in <module>
   F=open('pyton.txt',encoding="utf-8" )# снова открываем файл
FileNotFoundError: [Errno 2] No such file or directory: 'gaik.txt'
```

(нет такого файла или директории: 'gaik.txt').

А если вы допустили распространённую ошибку, записав:

```
A=open(garik.txt)
```

(Имя файла записывается в кавычках, или апострофах, или вообще в виде строчной переменной – а здесь этого нет!).

В таком случае на экране:

```
Traceback (most recent call last):
File "C:/Как бы диск D/Питоновы программы/open.py", line 1, in<module>
F=open(garik.txt)
NameError: name 'test' is not defined
```
(имя 'test' не определено)

И вот вам на основе этих проблем следующее упражнение:

Упражнение 19.2.3

Пользуясь ранее изученным оператором *try*, и программой поиска слова "питон" в 100 000 символов из предыдущего упражнения, смоделируйте обе возможные проблемы с открытием файла и сообщите пользователю о возникших проблемах. Если проблем нет – пересчитаем слова "питон" в файле.

Давайте, в конце концов, на основе имеющихся знаний создадим ещё одну полезную программу. Как какая ещё была полезная программа?! Программа для решения квадратных уравнений! Куда уж полезней!

Но это мы помогали математикам. Теперь – поможем самим себе!

Упражнение 19.3

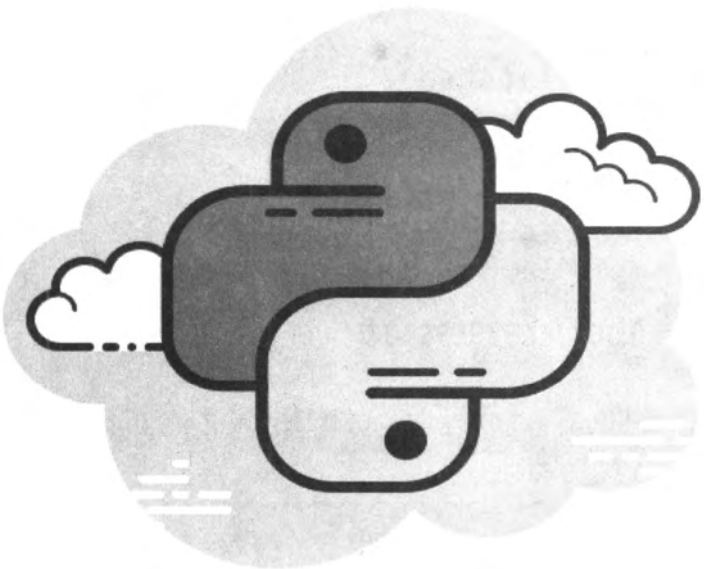
Надо написать программу "Телефонный справочник". Пусть в ней будет только фамилия, имя и телефон абонентов. Программа должна ловко находить в списке нужного абонента по его фамилии, или имени, или номеру. Не нашёлся нужный человек – предлагаем его ввести, и затем, естественно, сохраняем новый список на диске. Ещё пусть программа умеет выводить список на экран, да не как попало, а красиво – столбиками. Да, надо придумать "защиту от дурака" – не хорошо это, когда при малейшей ошибке клиента программа "вылетает"! Ну и вишенка на торте: в нашем случае программа должна позволять клиенту вводить № абонента как попало: и так 8(888) 000.0007 и эдак +7-888-000,0007 (хранить-то всё равно будем в виде 88880000007).

Да, **основная задача программирования**, как правило – получить какие-то данные, обработать их и сохранить на своём компьютере, передать по Интернету другому программисту, отправить на хранение в облако (нужное подчеркнуть). Работа несложная – открываем файл, обрабатываем, записываем и закрываем.

При создании программ нам частенько приходилось сталкиваться (и все столкновения заканчивались нашей победой, заметьте!) с необходимостью сделать какие-то однотипные преобразования во всех элементах последовательности. Как вы заметили, у добрых Творцов Питона почти на все наши выдумки есть готовые ответы, как правило, принимающие облик функций.

Глава 20.

НУ-КА ВСЕ ВМЕСТЕ



Допустим, надо возвести в квадрат все целые числа от 0 до 1 000 000, а результаты записать в какой-нибудь список. Да, и надо провести хронометраж этой работы. Да не "хренометраж", а узнать, сколько времени выполняется эта работа. Ну, теперь-то нам, наученным сладким опытом программирования, как говорят программисты – " Это как два пальц... Нет-нет! Как два байта переслать!":

```
from time import time
M1000000=range(1_000_000)
T=time()
A2=[]
for i in M1000000:
    A2.append (i**2)
print ('надо', time()-T,'секунд')
```

Результат:

```
надо 0.08774685859680176 секунд
```

Что случилось? Чему вы так удивлены? Первому оператору? Согласен, чудны дела твои, господи...[1]

Дело в том, что `range()` – это функция, и результатом её работы является ... ну конечно же, объект (который, как вы знаете, есть вещь в себе и обычными способами не разделывается)!

1 Библейский фразеологизм

Но мы имеем право назвать этот объект, например M1000000 и разобрать его на кусочки в цикле, что мы и сделали.

Итого хронометраж дал результат выполнения всей работы ~0,087 секунд.

Так вот, для такого рода работ (проделать какие-то занудные преобразования со всеми членами последовательности) создатели Питона придумали функцию **map()**:

```
map (<функция>, <любое количество последовательностей>)
```

причём функция здесь может быть любая – встроенная, пользовательская и даже лямбда-функция, причём записывать задаваемые функции надо **без скобок**.

Это значит, что с **map()** можно использовать любую функцию при условии, что эта функция принимает 1 аргумент и возвращает 1 значение.

Другие функции можно использовать лишь по желанию Питона: иногда съест, а иной раз и выплюнет.

Последовательности *– это списки, строки, кортежи и множества.*

Чтобы не быть голословным, проделаем ту же работу, что и раньше, с помощью функции **map()** (её название происходит от английского *mapping* – отображение):

```
M1000000=range(1_000_000)
T=time()
A2=[]
A2=map(Sqr,M1000000)
print ('map',time()-T)
```

Время выполнения:

```
map 0.007904529571533203
```

то есть на порядок меньше!

А вот возвращает **map()** некий объект, из которого правда, прекрасно получается шашлы…, (извините, я ещё не обедал), список.

Вот только при изготовлении из **map()** списка с помощью функции **list()** всё преимущество во времени съедается преобразованием:

```
M1000000=range(1_000_000)
T=time()
A2=[]
A2=list(map(Sqr,M1000000))
print ('map',time()-T)
```

и на экране

```
map 0.08194452395572151243
```

Тогда зачем вообще вся заморочка? Во-первых, использование функции **map()** экономит память. А во-вторых, решение с использованием **map()** немного убыстряет написание программы, ох, извините, кода – писать приходится на 2 строчки меньше. А в-третьих – это просто выглядит красиво и элегантно!

Вот давайте немного потренируемся в написании элегантного кода с применением функции map() в интерактивном режиме IDLE. Пусть у нас есть небольшой список:

```
>>>A123=[-0.123,1.23,12.345,123.456]
```

Сделаем все его члены целыми числами:

```
>>>Aint=list((map(int,A123)))
>>>Aint
[0, 1, 12, 123]
```

Неплохо!

Преобразуем все элементы в строчки:

```
>>>Astr=list(map(str,A123))
>>>Astr
['-0.123', '1.23', '12.345', '123.456']
```

Замечательно!

А если я захочу вычислить косинус от всех элементов A123? Нет проблем, если только не забыть импортировать функцию **cos()**:

```
>>>from math import cos
>>>Acos=list(map(sin,A123))
>>>Acos
[0.9924450321351935, 0.3342377271245026, 0.9755974241168766, -0.5947139710921574]
```

Прекрасно! А теперь давайте применим **map()** к пользовательской функции. К какой? А давайте извлечём корень 4-ой степени из всех элементов.

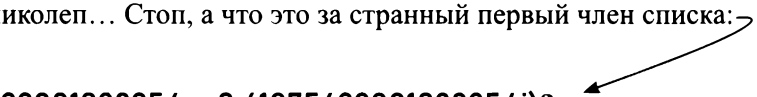
Для этой цели удобнее написать небольшую отдельную программку:

```
def SQRT(x):
    return x**0.25
A123=[-0.123,1.23,12.345,123.456]
ASQ=list(map(SQRT,A123))
print (ASQ)
```

На экране:

[(0.41875622881039254 + 0.41875622881039254j), 1.0531161619882878, 1.8744453094588205, 3.3333279999872]

Просто великолеп… Стоп, а что это за странный первый член списка:



(0.41875622881039254 + 0.41875622881039254j)?

А это я не учёл, что первый член исходного списка меньше нуля, но Питон отважно преодолел эту трудность, представив результат в виде *комплексного* числа(!). Да, Питон преспокойно работает и с комплексными числами. Правда, непонятно, почему вместо нормального обозначения мнимой единицы **i** [2] применяется обозначение **j**? Здесь, как всегда, "От вопросов скрыл ответы лес…"[3].

Надо исправить это недоразумение – ещё нам комплексных чисел не хватало! Усложним пользовательскую функцию – не будем извлекать корни из отрицательных чисел, а будем из них… ну, допустим вычитать моё любимое число 13:

```
def SQRT(x):
    if x>=0:
        return x**0.25
    return x-13
A123=[-0.123,1.23,12.345,123.456]
ASQ=list(map(SQRT,A123))
print (ASQ)
```

На экране:

```
[-13.123, 1.0531161619882878, 1.8744453094588205, 3.3333279999872]
```

Упражнение 20.1

С помощью лямбда-функции возведите в квадрат все члены последовательности [0,2,4,6,8,10,12,14,16]. Распечатайте результат. А теперь все члены последовательности [-3,0,3,6,9] возведите в куб и распечатайте, причём сделать это надо одной строкой.

2 Мнимая единица i обозначает число, которое при возведении в квадрат даёт -1. Знаю, что так не бывает, но вот некоторым математикам очень захотелось, и в результате родилась теория функции комплексного переменного, имеющая применения, как это ни странно, во вполне реальной жизни, в частности, в электротехнике

3 Это из замечательной песни А. Макаревича "Три окна"

А как **map()** работает с несколькими последовательностями? А вот так:

```
def Inst (a, b):
    return a**b
A0 = [9,11,3,4,11]
B2B = [0.5,2,5,-2,11]
print (list (map (Inst, A0, B2)))
```

То есть все числа из списка A0 последовательно возвели в степени из списка B2B. Получили:

```
[3.0, 121, 243, 0.0625, 285311670611]
```

А теперь сами:

Упражнение 20.2

Пусть у нас есть 4 последовательности:

A = [1, 2, 3, 4, 5]

B = (10, 9, 8, 7, 6)

C = [4.4, -2.2, 31.51, 20.6, 10]

D = [5.1, 2.2, 18.3, -2.12, 22]

Надо собрать из них пятую последовательность по формуле для n-ого члена $Q_n=A_n*B_n/(C_n+D_n)$.

Это у нас всё математика. А как дела со столь любимыми Питоном строками? Сейчас проверим!

Пусть у нас есть следующее высказывание:

```
>>>Chern='Хотели как лучше, а получилось как всегда'[4]
```

4 Наверное, многие позабыли автора этого бессмертного высказывания – В. Черномырдина, известного советского и российского политика

Легким движением превратим его в список строк (я понятно выражаюсь?):

```
>>>che=Chern.split()
>>>che
['Хотели', 'как', 'лучше,', 'а', 'получилось', 'как', 'всегда']
```

И применим к нему подходящий метод работы со строками (обращаю ваше внимание, что не любой метод работает с функцией **map()** – это довольно капризная дама).

Начнём с чего-нибудь попроще, давайте сосчитаем длину каждого элемента полученного списка:

```
>>>ChL=list(map(len,che))
>>>ChL
[6, 3, 6, 1, 10, 3, 6]
```

Мы сосчитали последовательно длины всех слов в великом высказывании знаменитого В. Черномырдина.

А теперь сделаем первые буквы заглавными:

```
>>>list(map(str.capitalize,che))
['Хотели', 'Как', 'Лучше,', 'А', 'Получилось', 'Как', 'Всегда']
```

Все буквы заглавными:

```
>>>list(map(str.upper,che))
['ХОТЕЛИ', 'КАК', 'ЛУЧШЕ,', 'А', 'ПОЛУЧИЛОСЬ', 'КАК', 'ВСЕГДА']
```

А ведь **print()** – это тоже функция! А ну-ка попробуем вот так:

```
>>>list(map(print,che))
```

Ура – сработало:

```
Хотели
как
лучше,
а
получилось
как
всегда
[None, None, None, None, None, None, None]
```

Вот только что означает последняя строчка? Здесь получилось, не как всегда: функция map() аккуратно передаёт в print() по одной значения из **Che**, печатает их и … ничего не возвращает в объект map(). А "ничего" по-английски "None". 7 строчек – и 7 раз получили **None** – вот они и выскочили на экран.

Ну ладно. Так ведь и input() – функция! А ну-ка, попробуем:

```
>>>Txt=['первое ','второе ','третье ']
>>>Menu=list(map(input,Txt))
```

На экране пойдёт бурная переписка:

```
первое борщ
второе котлета с рисом
третье компот
```

и результат

```
>>>Menu
['борщ', 'котлета с рисом', 'компот']
```

Обращу ваше внимание ещё на одно преимущество использования функции map() – список **Menu** не надо объявлять заранее – он организуется сам (точнее, его заводит и заполняет услужливая функция map()).

А теперь давайте, для тренировки, вычислим синусы вот такой линейки углов:

1,2,5,10,15,30,60,75,90

Углы даны в градусах, конечно. Коль речь у нас пошла о синусах, то надо вызвать модуль **math**. Всё-таки выражение "вызов модуля...", да ещё из каких-то таинственных глубин Питона, вызывает у меня сильные ассоциации с вызовом каких-то потусторонних сил из сказок. "Да пребудет с нами Сила[5] и модуль **math**!".

```
from math import *
```

Ну вот мы и вооружились и можем вступить в бой за святые синусы. Создадим сначала список градусов

```
grad=[1,2,5,10,15,30,60,75,90]
```

который надо превратить в список радианов (вы же помните, что синус работает только с радианами?), для чего удобно, опять-таки, использовать функцию **map**:

```
Rad=map(radians,grad)
print (Rad)
```

Проверим, как это работает (нажимаем F5):

```
<map object at 0x00000265F9196200>
```

Я забыл, что результат работы функции – объект – зверь опасный и непредсказуемый, работать с ним умеют только опытные программисты; ну а мы для укрощения используем проверенную функцию **list()**. Немного переделаем предпоследнюю строчку программы и запустим полученный код:

```
from math import *
grad=[1,2,5,10,15,30,60,75,90]
Rad=list(map(radians,grad))
print (Rad)
```

5 Ну тут уж все знают: это цитата из "Звездных войн"!

На экране теперь получим:

```
[0.017453292519943295, 0.03490658503988659,
0.08726646259971647, 0.17453292519943295, 0.2617993877991494,
0.5235987755982988, 1.0471975511965976, 1.3089969389957472,
1.5707963267948966]
```

Чудесные цифры, не так ли? Это радианы, полученные из градусов.

Ну теперь синусы получить легче лёгкого:

```
Sin=list(map(sin,Rad))
print (Sin)
```

и на экране:

```
[0.01745240643728351, 0.03489949670250097, 0.08715574274765817,
0.17364817766693033, 0.25881904510252074, 0.49999999999999994,
0.8660254037844386, 0.9659258262890683, 1.0]
```

но вот только разобраться, какой синус какому градусу соответствует, довольно сложно. Давайте создадим словарь, в котором будут градусы, а через двоеточие – соответствующие синусы.

Для этого используем промелькнувшую, как метеор в главе 17, функцию **zip()**. Это тоже гордая, объектообразующая функция, укрощаемая посредством функции **list()**:

```
ZIPPO=dict(list(zip(grad,Sin)))
print(ZIPPO)
```

Результат:

```
{1: 0.01745240643728351, 2: 0.03489949670250097, 5: 0.08715574274765817,
10: 0.17364817766693033, 15: 0.25881904510252074, 30: 0.49999999999999994, 60:
0.8660254037844386, 75: 0.9659258262890683, 90: 1.0}
```

Теперь намного понятней! Вот только углу в 30° соответствует значение синуса 0,49999… Зачем нам такая точность?

Попробуем округлить синусы посредством функции **map()**, причём в интерактивном режиме – для проверки:

```
>>> RndSin=list(map(round, Sin))
>>>RndSin
[0, 0, 0, 0, 0, 0, 1, 1, 1]
```

Ну конечно, ведь **round()** без параметров в скобках округляет до ближайшего целого числа!

А нам-то надо 2-3 цифры после запятой? Что же делать?

Давайте применим нашу краткую, как выстрел, лямбда-функцию:

```
from math import *
grad=[1,2,5,10,15,30,60,75,90]
Rad=list(map(radians,grad)) #градусы -> радианы
Sin=list(map(sin,Rad)) #радианы -> синусы          (21.1)
RndSin=list(map(lambda Rnd3: round(Rnd3, 3), Sin))
ZIPPO=dict(list(zip(grad,RndSin))) #сцепим градусы с синусами
print ('Градусы: значения синусов\n',ZIPPO)
```

И на экране всё стало красиво и понятно:

```
Градусы: значения синусов
{1: 0.017, 2: 0.035, 5: 0.087, 10: 0.174, 15: 0.259, 30: 0.5, 60: 0.866, 75: 0.966, 90: 1.0}
```

Спешу сообщить вам о довольно неожиданном открытии:

В начале главы я писал, что с **map()** можно использовать любую функцию при условии, что эта функция принимает 1 аргумент.

Так вот это не совсем верно…

Что вы говорите? "Аффтар, выпей йаду?". Ну что же, где-то вы правы…

А вот посмотрите: используя 2 последовательности, в функцию **round()** можно загнать второй аргумент.

В программу вместо строчки:

```
RndSin=list(map(lambda Rnd3: round(Rnd3, 3), Sin))
```

вставим вот такие 2 строки:

```
precision=(0,1,2,3,4,5,6,7,8)
RndSin=list(map(round, Sin, precision))
```

И в результате увидим на экране изумительный вывод:

```
Градусы: значения синусов
{1: 0.0, 2: 0.0, 5: 0.09, 10: 0.174, 15: 0.2588, 30: 0.5, 60: 0.866025, 75: 0.9659258, 90: 1.0}
```

То есть в кортеже **precision** мы задаём значения точности, и Питон послушно выдал значения синусов с нарастающим числом знаков после запятой!

Ну а теперь расскажу вам, за что ещё программисты любят функцию **map()**.

Во-первых, писать программу через **map()** удобнее, в чём вы сами уже убедились. И, во-вторых, если результаты работы **map()** не надо выводить на экран, то её частенько можно использовать и дальше прямо как объект в других функциях, так что нашу программу можно перезаписать в виде:

```
from math import *
grad=[1,2,5,10,15,30,60,75,90]
Rad=map(radians,grad)
Sin=map(sin,Rad)                                    (21.2)
RndSin=list(map(lambda Rrr: round(Rrr, 3), Sin))
ZIPPO=list(zip(grad,Sin))
print(ZIPPO)
```

И вот теперь-то она заработает раз в 100 быстрее!

Упражнение 20.3

Сравните скорость исполнения программы (21.1) с её же вариантом (21.2), а то мало ли что я тут понапридумывал.

Как нам уже известно, Питон умеет работать только с текстовыми файлами (ну, это не совсем верно, но пока будем считать так). Так вот, обогащённые знаниями о функции **map()**, давайте сделаем:

Упражнение 20.4.1

Следует создать текстовый файл из 10 000, естественно, случайных чисел, не превышающих 100 000, с двумя знаками после запятой. При создании следует применить функцию map(). Для чего это надо? Исключительно для того, чтобы сделать упражнение 20.4.2!

Упражнение 20.4.2

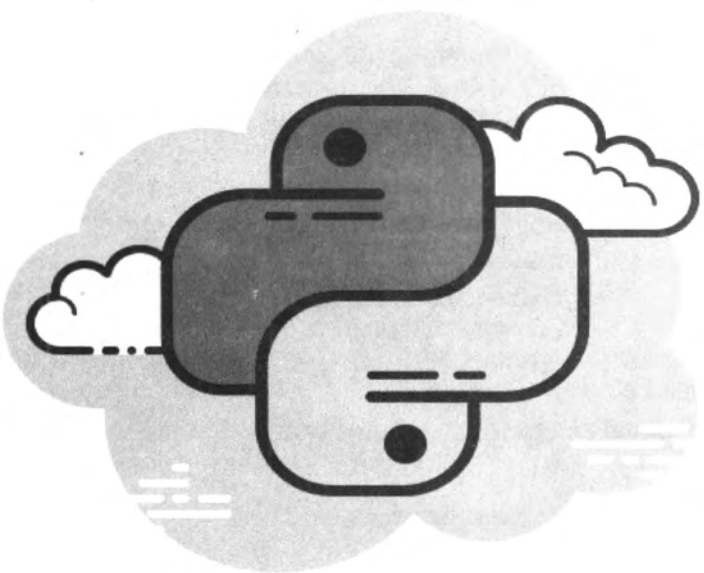
Следует открыть файл, созданный в предыдущем упражнении и найти в нём все пары чисел, сумма которых делится на число 131 313!

Функция **map()** оказалась не страшной и даже весьма полезной: избавляет нас от часто встречающихся довольно занудных преобразований и позволяет весьма лихо обрабатывать любые последовательности по заданным алгоритмам.

Что-то мы всё о серьёзном. Давайте немного подурачимся с пользой для окружающих!

Глава 21.

ВРЕМЯ ПОИГРАТЬ



Мы уже несколько раз использовали модуль **time**, который занимается измерением времени.

А давайте на его основе сделаем простенькие игры? Игры? С временем? Ну, не совсем игры! Скорее тесты. Прямо сразу и начнём!

```
import time
print("+++ сейчас мы измерим вашу реакцию +++")
```

Естественно, для начала импортируем модуль **time** и начнем готовиться к измерению скорости реакции пользователя. Для начала надо расслабить пользователя, то есть сделать задержку программы на какое-то время.

Модуль **time** любезно предоставляет нам метод, который так и называется **.sleep(t)** – и задерживает выполнение программы на **t** секунд (*sleep* по-английски "спать").

```
time.sleep(3)
st=time.time()
```

Засекаем время и ждём нажатия клавиши:

```
Txt = input ("Нажмите клавишу Enter!")
fin=time.time()
```

Далее вычисляем время реакции:

```
reac=fin-st
print("Время реакции: ", reac," секунд")
```

Запустите программу... Как там ваша реакция?

Ну в одну и ту же клавишу попадать достаточно просто.

Усложним задачу:

Упражнение 21.1

Пусть теперь пользователь нажимает на определённую числовую или буквенную клавишу, которая определяется случайным способом, а мы рассчитаем время реакции.

Упражнение 21.2

Напишите программу, которая проверяет ваше умение определять интервалы времени. Программа задаёт время, через которое надо нажать клавишу *Enter*, и выдаёт величину допущенной ошибки.

Это всё хорошо, вот только мерять время в секундах от начала каких-то незапамятных времён как-то не очень удобно, это почти как у поэта: "Какое, милые, у нас / Тысячелетье на дворе?"[1]

Нам бы поточнее желательно - в годах, днях и минутах... Пожалуйста!

Есть такой милый метод **.ctime()**. Если ему скормить количество секунд, прошедших с начала эпохи, то в ответ мы получим **строку**, содержащую местное время:

```
>>>sec = time.time()
```

[1] Это, разумеется, цитата из стихотворенья Б. Пастернака

```
>>>Ltime = time.ctime(sec)
>>>print(Ltime)
```

Вот она, строка:

```
Thu Jan 19 12:46:34 2023
```

Вот только строка на английском, увы…

А глава СЛОВАРИ зачем в этой книге? Вот давайте, воспользовавшись словарём, переделаем время и дату с английского на русский.

```
import time
sec = time.time() #получим число секунд
Ltime = time.ctime(sec) # Строка с местным временем
WEEK=Ltime[0:3] #день недели
MON=Ltime[4:7] #Месяц
DAY=Ltime[8:10] #Дата
TIME=Ltime[11:19] #Время
YEAR=Ltime[20:25] #Год
# Словарь переводит дни недели с английского на русский
WeRu={'Thu':"Четверг",'Fri':"Пятница",'Sat':
"Суббота",'Sun':"Воскресенье","Mon":"Понедельник",
'Tue':"Вторник",'Wed':"Среда"}
WRu=WeRu[WEEK] # заменяем англ. день недели на русский
# Словарь переводит названия месяцев на русский
MonRu={'Jan':"января",'Feb':"февраля", 'Mar':"марта",
'Apr':"апреля",'May':"мая",'Jun':"июня",'Jul':"июля", 'Aug'
:"августа",'Sep':"сентября",'Oct':"октября",'Nov':"ноября",
'Dec':"декабря"}
MRu=MonRu[MON] #заменяем англ. месяц на русский
print ('         В вашей местности сейчас')
print (DAY,MRu,YEAR, ' года, ',WRu)
print ('         Время', TIME)
```

На экране всё красиво:

```
В вашей местности сейчас
19 января 2023 года, Четверг
     Время 17:29:12
```

Ну а теперь, зная, как получить время и дату на данный момент и используя СЛОВАРЬ для перевода, сделаем:

Упражнение 21.3

Вы заметили, что в последнее время стало модно указывать, например, в каком возрасте Пеле[2] забил свой первый гол на чемпионате мира, причём возраст указывается в днях, месяцах и годах? Почему? Появились соответствующие программы.

Ну а мы чем хуже? Давайте напишем программу, которая, получив ваше время и место рождения - нет, место рождения, по-моему, не надо! – сообщит возраст вас, любимого, в днях, месяцах и годах. При этом я хочу, чтобы при выводе числительное было согласовано с существительным, то есть чтобы вывод был в виде:

Ваш возраст 1 год 4 месяца 29 дней

Или

Сегодня вам 11 лет 6 месяцев 22 дня

А не:

Ваш возраст 1 лет 4 месяцев 21 дней

Для согласования числительного с существительным можно использовать программу из упражнения 5.5!

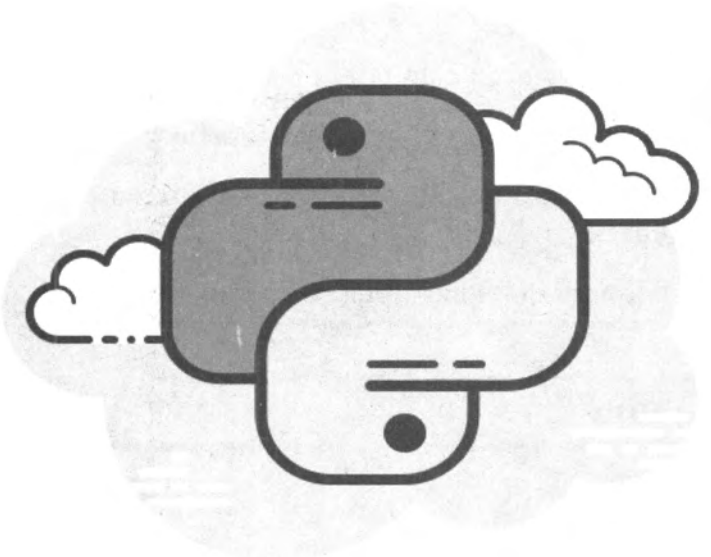
Ну что ж, книга прочитана, упражнения, надеюсь, сделаны. Я вынужден попрощаться со своими умными и, главное, терпеливыми и любознательными читателями. Надеюсь, вам было интересно. А ещё больше надеюсь, что моя книга помогла вам распахнуть дверь в интереснейший, занимательный, чрезвычайно обширный и богатый (во всех смыслах этого слова) мир программирования на Python!

DIXI[3]

2 Пеле – великий бразильский футболист, единственный на планете трёхкратный чемпион мира по футболу. Свой первый гол на чемпионатах мира он забил в возрасте 17 лет, 7 месяцев и 27 дней!

3 Я все сказал (латынь).

ОТВЕТЫ НА УПРАЖНЕНИЯ



Упражнение 0.1

Если считать по 1 числу в секунду по 8 часов в день, причём без выходных, то это займёт почти 35 дней!

Упражнение 1.1

Cat2y, Almaz1996, R____T, O600, – правильные имена переменных.

Неправильные имена:

Out! , TU-154 – содержат символы *!* и –;

3R, 3_m – начинаются с цифры;

_3m – начинается с символа подчёркивания;

DooШ – содержит букву Ш. Использовать можно, но не рекомендуется;

import – совпадает с ключевым именем.

Упражнение 2.1.

```
>>>x=2.5
>>>y=4
>>>x=x*y
10.0
>>>y=x-y
```

```
6.0
>>>y=x//2
5.0
>>>y=y%3
2.0
>>>x=x/3
3.3333333333333335
>>>x=x//1
3.0
```

Сразу, конечно, не получилось? Ну что поделаешь – "тяжело в учении – легко в бою"...[1] А, кстати, что за странная последняя операция x=x//1? Так как здесь *целочисленное деление*, то таким образом отброшена дробная часть. Вскоре вы узнаете, как это сделать красиво, а не столь корявым способом.

Упражнение 2.2.

Должно получиться вот так:

```
>>>a=4
>>>b=3
>>>a=(a+b)/2
3.5
>>>a=a**2-2*b
6.25
>>>b=b-a//(a-b)
2.0
>>>a=a**0.5
2.5
```

Да, последний оператор извлекает квадратный корень из 6,25 и получает 2,5.

1 Слегка переделанное высказывание великого полководца А. В. Суворова. На самом деле он сказал так: "Тяжело в учении — легко в походе! Легко в учении — тяжело в походе!"

Упражнение 3.0

Легко сказать – оцените! А как? Очень просто: надо заданную площадь разделить на 1 000 000 квадратиков. В центре каждого квадратика поместим горошину. Расстояние между горошинами – длина стороны полученного квадратика. Ну а по площади квадрата мы легко вычислим его сторону, для этого надо извлечь квадратный корень. Как извлекать корни (квадратные, конечно) – я рассказал в упражнении 2.2.

Вот вам и программа:

```
Sq=125
Sq=Sq*10000
q=Sq/1000000
L=sq**0.5
print ('Расстояние между центрами горошин было менее',
L, "см")
```

На экране:

Расстояние между центрами горошин было менее 1.118033988749895 см

То есть сами горошины должны быть меньше 1 см! А ведь комната в 125 квадратных метров побольше хорошей квартиры! Да, миллион – это очень большое число...

Упражнение 3.1

Нет, ребята, так:

```
x=3
y=2
```

дело не пойдёт! Представьте, что вам надо поменять значения x и y где-то в середине программы, а реальные значения x и y вы не знаете.

Ну и что тут сложного? Тогда вот так:

```
x=2
```

```
y=3
x=y
y=x
print ('x=',x,'y=',y)
```

Набираем – запускаем – с изумлением читаем:

```
x= 3 y= 3
```

После выполнения оператора `x=y` прежнее значение 2 из `x` исчезло, `x` стал равен `3` и `y` благополучно получил назад своё же значение 3… Что же делать? Вводить вспомогательную переменную:

```
x=2
y=3
w=x
x=y
y=w
print ('x=',x,'y=',y)
```

Вот теперь правильно, на экране читаем:

```
x= 3 y= 2
```

Небольшой фокус! В этом изумительном Питоне есть понятие группового присваивания, то есть без вспомогательной переменной:

```
x=2
y=3
x,y=y,x
print ('x=',x,'y=',y)
```

На экране:

```
x=3 y=2
```

Пока не рекомендую пользоваться этим фокусом: во-первых, это ухудшает чтение программы, а во-вторых, там есть

подводные камни при работе со списками. Что такое списки? "Всякому овощу своё время" – узнаете в соответствующей главе.

Упражнение 3.2

Возьмём `T` сразу побольше:

```
T=10000
```

Количество минут определяется запросто, как остаток от деления T на 60:

```
Min=T%60
```

А теперь разделим T на 60 нацело, чтобы получить количество часов:

```
T1=T//60
print (T1)
```

Запустим полученную программу, чтобы посмотреть, что у нас получилось (кстати, так и следует отлаживать программы – написали небольшой, но совершенно понятный кусочек большой программы – запустили, посмотрели на результат):

```
166
```

М-да, что-то многовато получилось – в сутках же всего 24 часа… О! 24 часа! Да надо же просто `T1` разделить нацело на 24 и полученный результат вычесть из T1! Ой… Да это же остаток от деления:

```
Hour=T1%24
print (Hour)
```

На экране:

```
166
22
```

Вполне разумный результат. Вот теперь убираем из программы лишнее, задаём для проверки `T`... А к чему бы нам приравнять `T`, чтобы можно было легко проверить результат? Пусть для начала `T=300` минут, то есть 5 часов.

Набиваем программу, убирая из неё лишние операторы `print()`, которые мы вводили для отладки:

```
T=300
Min=T%60
T1=T//60
Hour=T1%24
print (Hour,':',Min)
```

Оператором `print()` можно печатать несколько переменных, отделяя их друг от друга запятой.

На экране – показания электронных часов:

```
5 : 0
```

Согласен, часики у нас довольно неказистые... А вот как сделать их вполне "казистыми" мы узнаем немного позже, в главе "ЕЩЁ АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ".

А теперь поставьте `T=10000` и не забудьте полюбоваться на результат:

```
22 : 40
```

Упражнение 3.3

Ну конечно же, вы уже написали что-то подобное:

```
R=7
A=60
Omega=(A/R)**0.5
print ('Скорость вращения:',Omega)
```

Переменная A равна 60 м/сек² – это перегрузка 6g – ускорение свободного падения составляет почти 10 м/сек². Запускаем, мгновенно получаем ответ:

Скорость вращения: 2.9277002188455996

И только теперь задумываемся о единицах измерения. Скорость вращения ω в чём измеряется? Отвечаю – в радианах в секунду, как и положено в математике… Придётся радианы в секунду пересчитать в обороты в минуту!

2π радианов в секунду – это 1 оборот в секунду, или 60 оборотов в минуту. Следовательно, для пересчёта надо применить формулу:

$$N=\frac{\omega}{2\pi}\cdot 60$$

И немного переписать программу:

```
R=7
A=60
Pi=3.14
Omega=(A/R)**0.5
N=Omega*30/Pi
print ('Скорость вращения:',N, 'оборотов в минуту')
```

На экране:

Скорость вращения: 27.971658141836937 оборотов в минуту

Ну вот, совсем другое дело! Если посмотреть фильм о подготовке Гагарина к полёту, то можно увидеть, что центрифуга где-то так и вращается – со скоростью пол-оборота в секунду.

Упражнение 3.4

Примерно так:

```
L=42195
```

```
H=2
M=10
```

В переменную *L* запишем длину дистанции в метрах, *H* и *M* – это время в часах и минутах:

```
sec=H*3600+M*60
```

Пересчитали время на секунды:

```
V=L/sec
```

Рассчитаем скорость марафонца в метрах в секунду:

```
t=1000/V
```

За время `t` (в секундах) бегун пробежит 1000 метров:

```
tS=t%60
tM=t//60
```

Время в секундах получим как остаток от деления t на 60, а время в минутах – результат деления нацело.

```
print ('марафонец пробегает 1 км за:',tM,'минуты и',
tS,'секунд')
```

Остаётся только вывести результат на экран в понятной форме и ... застыть в страшном изумлении:

```
марафонец пробегает 1 км за: 3.0 минуты и 4.856025595449694
секунд
```

оказывается, марафонец-чемпион проносится КАЖДЫЙ километр своей дистанции быстрее норматива III взрослого разряда по бегу на 1 км!

Упражнение 4.1

Да запросто:

```
prob=' '
star='*'
print (star*21)
print(star,prob*17,star)
print(star,prob*7,'+',prob*7,star)
print(star,prob*17,star)
print (star*21)
```

Упражнение 4.2

"Да это легче лёгкого! – произнёс Эмиль с некоторой досадой"[2]. Простейшая программа, не правда ли:

```
Rows=15
Seats=22
```

Номер фильтра, где находится послание:

```
N=174
```

Рассчитываем номер места как остаток от деления номера фильтра на число мест:

```
Seat=N%Seats
```

А ряд вычислим, просто разделив нацело номер на число мест:

```
Row=N//Seats
print ("Ряд",Row,', место',Seat)
```

На экране видим:

```
Ряд 7, место 20
```

2 Цитата из книги А. Линдгрен "Эмиль из Лённеберги"

Шик, блеск, красота! Ага, ну не совсем шик-блеск… Для начала я бы всё же проверил работу программы на тестовых задачах. Ну например, пусть N равно 1, естественно. Ожидаемый ответ "Ряд 1, место 1". На экране:

Ряд 0 , место 1

Вот это да!... А если N=22?

Ряд 1 , место 0

Кажется, Юстас наше донесение не получит… Что делать? Где ошибка?

Как бы было хорошо, если б нумерация рядов и мест начиналась с 0! Ага! Значит к номеру ряда и места надо добавить по единичке! Заменим 3 и 4 строчки в программе:

```
Seat=N%(Seats)+1
Row=N//(Seats)+1
```

Полагаем N=22:

```
Ряд 2, место 1
```

А если N=1:

```
Ряд 1, место 2
```

Не помогло… Думаем дальше…

Стоп! Эврика!!![3] Да надо сам номер N уменьшить на 1, перед началом расчётов. Тогда имеем:

```
Rows=15
Seats=22
N=66
N=N-1
Seat=N%(Seats)+1
Row=N//(Seats)+1

print ("Ряд",Row,', место',Seat)
```

[3] Легендарное восклицание Архимеда, выражает радость при разрешении трудной задачи

Проверяем, при N=1:

```
Ряд 1, место 1
```

При N=22:

```
Ряд 1, место 22
```

Ну а при N=23:

```
Ряд 2, место 1
```

И наконец при N=174 имеем:

```
Ряд 8, место 20
```

Ну вот, теперь можно и донесение отправлять...

Заодно мы раскрыли великую тайну программистов: почему нумерация последовательностей начинается с 0? А потому, что если бы нумерация рядов и мест и фильтров у нас начиналась бы с нуля, то в таком случае наш самый первый вариант программы сработал бы совершенно спокойно и правильно!

Упражнение 4.3.

Можно так:

```
Name=input (' Как вас зовут? ')
Y=" Привет, "
Y=Y+Name+'!'
print (Y)
```

а можно и так:

```
Name = input (' Как вас зовут? ')

print (" Привет, ", Name, '!')
```

Упражнение 4.4.

Здесь всё просто:

```
Fam=input("Фамилия ")
Name=input("Имя ")
Year=input("год рождения ")
print('Здравствуйте,',Name, ' ' ,Fam)
HowOld=2023-int(Year)
print('Вам',HowOld,'лет')
```

Тут самое главное – не забыть переменную `Year` из текстовой превратить в число. Питон всё, что вводится, обрабатывает как текст – повторяю ещё раз для тех, кто забыл в этой программе поставить функцию `int()` перед `Year` и теперь задумчиво смотрит на экран, где написано:

```
HowOld=2023-Year
TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

Перевожу с питонного на человеческий последнюю строчку: "Человек! Ну опять из числа просишь вычесть слово? Ну и что получится, если из 10 вычесть троллейбус?!" Ой, я, кажется, вместо перевода сделал пересказ, хотя от этого смысл даже усилился. Вместо:

```
HowOld=2023-Year
```

вставьте:

```
HowOld=2023-int(Year)
```

и будет вам счастье!

Упражнение 4.5.

```
rost=int(input("Ввѣдитѣ ростъ человѣка въ вѣршкахъ:"))
print ("Рост этого человека" ,rost*4.44+2*71.11, "см")
```

Рост Герасима составлял 195,5 см – настоящий гигант, если учесть, что средний рост мужчины в то время был 161 см!

Ещё раз обращаю ваше внимание на то, что в качестве разделителя в числах используется *точка*. Если вы по ошибке поставите *запятые* (напи́шите `rost*4,44+2*71,11`) то получите совершенно неожиданный результат:

Рост этого человека 48 186 11 см

Заметили, что в операторе `input` вставлена запись в дореволюционной орфографии? Да, Питон совершенно толерантен к любому языку...

Упражнение 4.6.

```
>>>'{:013.3f}'.format (2023)
'000002023.000'
```

Упражнение 4.7.

Программа простая, но длинная.

Сначала зададим диаметр мяча:

```
ball=22
```

У нас все входные данные о Солнце и планетах в километрах, а надо в сантиметрах. Вот переменная `coeff` и показывает, сколько в километре сантиметров.

```
coeff=1000*100
Sun=1.39e6 * coeff
Earth=12742 * coeff
Jupiter=142984 * coeff
```

Расстояние от Солнца до Земли:

```
R_S_E=1.49e8 * coeff
```

... и Нептуна

```
R_S_N=4.55e9 * coeff
```

Масштаб уменьшения Солнечной системы:

```
M=Sun/ball
print ('Масштаб','{:.1E}'.format(M))
dEarth=round(Earth/M,2)
print ('Диаметр Земли теперь:',dEarth,'см')
dJupi=round(Jupiter/M,2)
print ('Диаметр Юпитера теперь:',dJupi,'см')
rSE=round(R_S_E/M)
print ('Расстояние до Земли теперь:',rSE,'см')
rSN=round (R_S_N/M)
print ('Расстояние до Нептуна теперь:',rSN,'см')
```

Результат, конечно, поражает:

Масштаб 6.3E+09
Диаметр Земли теперь: 0.2 см
Диаметр Юпитера теперь: 2.26 см
Расстояние до Земли теперь: 2358 см
Расстояние до Нептуна теперь: 72014 см

Земля превратилась в 2-хмиллиметровую песчинку в 23 метрах от Солнца, а Нептун оказался в 720 метрах от футбольного мяча, изображающего Солнце!..

Упражнение 5.1

```
print (2==3)
print (0.001<0.001)
```

Ответы: False и False

```
Logic=True
print (Logic)
```

На экране:

```
True
```

Питон выводит на экран *значение* переменной Logic, которое в данный момент – *True*.

А теперь похитрее:

```
L112a=(2>3)
print (L112a)
```

Видим:

```
False
```

Здесь мы присвоили переменной `L112a` значение логического выражения `2>3` и Питон вполне логично напечатал False.

Упражнение 5.2

Возьмём стандартное квадратное (интересно, а круглые и шестиугольные уравнения бывают?) уравнение:

$ax^2 + bx + c = 0$

Вводим коэффициенты (с подсказкой):

```
print ('введите коэффициенты квадратного уравнения ax²+bx+c')
a=int(input("a="))
b=int(input("b="))
c=int(input("c="))
```

Вычисляем дискриминант:

```
D=(b**2-4*a*c)
```

Если дискриминант меньше 0, то корней нет:

```
if D<0:
      print (" Корней нет! ")
else:
```

Если дискриминант больше 0, то находим корни по формуле:

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$$

```
x1=(-b+D**0.5)/(2*a)
x2=(-b-D**0.5)/2/a
```

(я привёл здесь 2 способа, как записать деление на 2a)

```
print (x1, x2)
```

Ну вот теперь – "Спасибо нам скажет сердечное школьный народ...[4]"

Упражнение 5.3

Ответ:

```
x=int(input(" Введите целое число: "))
if x%2==0:
    print (" чёт ")
else:
    print (" нечёт ")
```

Но можно ещё проще и красивее:

```
z=int(input('Введите целое число: '))
Str="чёт"
if z%2!=0:
    Str='нечёт'
print (Str)
```

Упражнение 5.4

Программка, конечно, проще некуда, зато весьма занудная. Самое трудное в ней – это набрать названия всех 7 дней недели.

4 Здесь автор позволил чуть-чуть перефразировать Н. Некрасова:
"Сейте разумное, доброе, вечное,
Сейте! Спасибо вам скажет сердечное
Русский народ..."

```
day=int(input('Введите количество дней, прошедших с
начала года:'))
Dweek=day%7
if Dweek==1:
    print('Воскресенье')
elif Dweek==2:
    print('Понедельник')
elif Dweek==3:
    print('Вторник')
elif Dweek==4:
    print('Среда')
elif Dweek==5:
    print('Четверг')
elif Dweek==6:
    print('Пятница')
else:
    print('Суббота')
```

Комментировать в этой программе в общем-то нечего...

Упражнение 5.5

Для начала проверим сочетаемость слова "рубль" с различными числительными. 1 рубль; 2,3,4 – рубля; 5,6 – рублей; 10,11,12 – всё равно "рублей"; 20 рублей; 21 – рубль; 22,23,24 – рубля и дальше 25 рублей. В общем, алгоритм понятен. Составляем программу:

```
rub=int(input('Сколько у вас рублей? '))
```

Пользователь вводит число рублей.

```
if 10<=rub<=20 or rub%10>4:
    txt='рублей'
```

От 10 до 20 пишем "рублей", и если остаток от деления числа рублей на 10 больше 4 – тоже "рублей":

```
elif rub%10==1:
```

```
    txt='рубль'
```

1 – "рубль", и 21 – "рубль", и … 41 – "рубль"

```
else:
    txt='рубля'
```

2,3,4 – пишем "рубля":

```
print ("У вас ",rub,txt)
```

Запускаем, проверяем: 1, 2, 3, …, 10, 11, 12, …, 20, 21, 22, 25, 26 – всё верно! Ура! Проверим 30! …Бумс! "У вас 30 рубля"… Программа пошла по нерасчётной траектории… Вот так оно и бывает – немного ошиблись в алгоритме. Понятно, в чём дело – мы не учли случай, когда при делении на 10 остаток будет 0. Тогда надо 2-ую строку программы записать в виде:

```
if 10<=rub<=20 or rub%10>4 or rub%10==0:
```

Запускаем-проверяем исправленную программу – всё чудесно: 30 рублей, 50 рублей… А если 100? Всё правильно: 100 "рублей"! Ура!

Стоп! "…Меня терзают смутные подозрения…[5]". А если 111 рублей? Так и есть – ответ: "У вас 111 рубль". А почему? А потому, что во 2-ой строке у нас в списке исключений только числа от 10 до 20, а тут – 111 рублей! А с числами меньше 100 программа работала правильно… Значит, проще всего в числах, которые больше 100 отбрасывать 100. Немного переделаем программу:

```
Rub=int(input('Сколько у вас рублей? '))
rub=Rub%100
```

Этой операцией отбросим сотни (а также тысячи), а дальше всё по-старому:

```
if 10<=rub<=20 or rub%10>4 or rub%10==0:
    txt='рублей'
```

5 Ну конечно же, из бессмертной комедии Л. Гайдая "Иван Васильевич меняет профессию"

```
elif rub%10==1:
    txt='рубль'
else:
    txt='рубля'
print ("У вас ",Rub,txt)
```

А вот на экран выводим *введённое* число. Теперь программа работает корректно!

Упражнение 6.1

Предлагаю программу вот в таком виде (ваша программа вполне может отличаться от моей, главное – чтобы работала правильно):

```
print ('Вас приветствует портал "СКАЗОЧНЫЙ МИР"')
txt='вы не из нашей сказки'
Name=input('Ваше имя:')
if (Name=='Илья') or (Name=='Алёша' or Name=='Алеша' )
or (Name=='Добрыня'):
    Surname=input('Ваше прозвище:')
    Ilya=(Surname=='Муромец' and Name=='Илья')
    Alesha=(Surname=='Попович' and ((Name=='Алёша')
or (Name=='Алеша')))
    Dobryn=(Surname=='Никитич' and Name=='Добрыня')
    if Ilya or Alesha or Dobryn:
        txt='Добро пожаловать!'
print (txt)
```

Сначала выводим приветствие, а в переменную *txt* заранее загружаем текст "вы не из нашей сказки". Просим ввести имя и проверяем его на правильность, причём для Алёши вариант "Алеша" тоже будем считать правильным. Если пользователь ввёл верное имя, просим ввести прозвище[6], а дальше... Если сверять в операторе *if* каждого из 3 былинных богатырей по сочетанию имя-прозвище, то получится очень длинно и запутано. А запутанности в программировании надо бояться

6 До XII века фамилий на Руси ещё не было. Были имена и прозвища по роду деятельности, месту происхождения или какой-то другой особенности предка

как огня[7]!!! Поэтому я ввёл 3 логические (!) переменные. Если доступа просит богатырь с правильным именем и прозвищем, то одна из этих переменных примет значение *True*, переменная *txt* будет переименована и вход будет открыт.

Проверяем: "Илья Муромец", "Алеша Попович" – добро пожаловать, а вот "Иван", "Илья Попович" – не из нашей сказки.

Упражнение 6.2

```
print (2>3 and 1<=4)
print (2>3 or 1<=4)
```

Здесь всё просто:

```
False
True
```

А теперь?

```
a=2<3
print (not a)
b=a and True
print (b)
c=2>3
e=a or b and c
print (e)
```

На экране:

```
False
True
True
```

7 В истории программирования было время, когда при написании программ очень широко применялся оператор goto, позволяющий перепрыгивать в любое место программы, возвращаться назад. Все это приводило к тому, что код программы становился страшно запутанным и похожим на тарелку со спагетти, и было очень трудно понять, что и как программа делает. В 1970 году Н. Вирт разработал язык Паскаль, из которого этот ужасный оператор впервые был изгнан. Результат? Программы стали понятней, а скорость отладки выросла!

Упражнение 6.3.

"Подумаешь, бином Ньютона[8]":

```
for X in (False,True):
    for Y in (False,True):
        A=X and not(Y) or not(X) and Y
        print (X,'XOR',Y,'  = ',A)
```

А что за странное слово `XOR`, вставленное в оператор `print`? Дело в том, что мы делаем таблицу истинности для логической операции, исключающей "или", которая записывается как XOR, а в математике обозначается как $\oplus$.

На экране:

```
False XOR False   =  False
False XOR True    =  True
True XOR False    =  True
True XOR True    =  False
```

Упражнение 6.4

Программа совершенно несложная:

```
a=float(input('Первое число: '))
b=float(input('Второе число: '))
if (a>10 or a<5) and (b>10 or b<5):
    print ('числа подходят')
elif(a>10 or a<5) or (b>10 or b<5):
    print ('подходит только 1 число')
else:
    print ('числа НЕ подходят')
```

Здесь самое главное – не запутаться в логических операторах и аккуратно и точно расставить скобки.

8 Фраза из великого романа "Мастер и Маргарита" М. Булгакова. Так говорят о какой-либо простой, несложной задаче, которую кто-то не может решить, полагая её сверхсложной

Упражнение 6.5

Тут, как всегда, главное – аккуратно перевести выражение ¬ ((X>2) → (X>3)) с математического языка на программистский, а дальше уравнение решается мгновенно:

```
for x in range(10):
    if not ((x>2)<=(x>3)):
            print(x)
```

на экране – решение уравнения:

```
3
```

Упражнение 6.6

Аккуратно превращаем математическое выражение в выражение на языке Питон, а в цикле ставим верные значения для заданного отрезка [20; 24]:

```
for x in range(20,25):
    if ((x<25)<=(x<23)) and ((x<22)<=(x>21)) :
           print(x)
```

и читаем ответ:

```
22
```

Упражнение 6.7

Из обозначения]40; 60[видно, что конечные точки не входят в заданный промежуток, поэтому программа выглядит так:

```
for x in range(41,60):
    if not((x%5==0)<=(x%25==0)):
        print(x, end=' ')
```

а на экране:

```
45 55
```

Упражнение 7.1

Например, можно вот так:

```
Summ=0
Finish=55
STEP=7
for i in range (13, Finish+1, STEP):
    print (i, end='')
    if i! = Finish:
        print ('+', end='')
    Summ+=i
print (' =', Summ)
```

Комментарии: сначала объявляем сумму последовательности равной 0. Переменная `i` перебирает все значения, начиная с 13 и кончая 55 с шагом 7 (чтобы захватить и значение 55, я к значению `Finish` добавил 1). Эти значения выводим на экран и не переходим при этом на следующую строку (параметр end=' '). Дальше оператором `if` проверяем, не дошли ли мы до последнего числа. Если не дошли, то печатаем '+' и увеличиваем сумму на значение `i` (предпоследняя строка). Выйдя из цикла, печатаем знак равенства и итоговую сумму:

```
13+20+27+34+41+48+55 = 238
```

Упражнение 7.2

```
for i in (0, 4, 7, 2,1,19,6):
    for j in (18,1,9, 3, 4):
        if i==j:
            print (i, end=" ")
```

Пробелы на разных уровнях обязательны! На экране, разумеется:

```
4 1
```

Упражнение 7.3

Ну, например вот так:

```
for Dd in (-2, 5, 8, -4, -6, -11):
   if Dd>0:
        print (Dd)
   else:
        Dd=-Dd
        print (Dd)
```

А лучше вот так:

```
for Dd in (-2, 5, 8, -4, -6, -11):
    if Dd<0:
        Dd=-Dd
    print (Dd)
```

Как я писал выше, оператор `else` не обязателен, чем я и воспользовался.

Упражнение 7.4

Ответ: 243

Упражнение 7.5

```
start=True
while start:
    x = input("Пароль: ")
    if x=='123':
       start=False
    else:
        print ('Неверный пароль ')
print ('Вход разрешён')
```

Обратите внимание, что в самом начале программы я

вынужден присвоить `True` индикатору `start`, чтобы цикл `while` начал работу (если `start` присвоить `False`, то заголовок цикла `while` проверит логическое условие в переменной `start`, обнаружит, что оно ложно и ... цикл не будет выполнен ни разу, да ещё и вход будет разрешён).

Можно ещё немного сократить программу, поставив проверку и ввод пароля прямо в заголовок цикла (Питон на удивление лаконичный язык). Короче некуда:

```
while input("Пароль: ")!='123':
      print ('Неверный пароль ')
print ('Вход разрешён')
```

Здесь введённый оператором `input` пароль, тут же, в заголовке цикла, сравнивается с правильным паролем. Если пароль неверен, логическое выражение в цикле `while` принимает значение *True* (ведь введённое значение НЕ равно 123!), печатается "Неверный пароль" и цикл повторяется. Если пароль правильный, логическое выражение в цикле становится `False`, цикл прерывается и вход разрешён.

Упражнение 7.6

"Элементарно, Ватсон[9]":

```
N=0
while True:
    N+=2
    if N==6:
        continue
    print (N, end=' ')
    if N==20:
        break
print ("\n我会回来的")
```

9 Эта знаменитая фраза ни разу не встречается в книгах Конан Дойля! А пришла она к нам, как водится, из кино, а именно из советского сериала "Приключения Шерлока Холмса и доктора Ватсона"

и на экране:

2 4 8 10 12 14 16 18 20
我会回来的

Что? Как перевести фразу на китайский? Элементарно: заходите в любой переводчик в Интернете и просите его перевести фразу *I'll be back* на китайский... Потом копируете полученный перевод и аккуратно вставляете его в оператор `print`.

Обратите внимание: иероглиф 我 – это "я" по-китайски – очень затейливое написание!

Упражнение 7.7

```
for L in range(10):
    for I in range(10):
        for K in range(10):
            for B in range(10):
                for U in range(1,10):
                    if L!=I and L!=K and L!=B and L!=U:
                        if I!=K and I!=B and I!=U:
                            if K!=B and K!=U:
                                if B!=U:
                                    LIK=L*100+I*10+K
                                    BUBLIK=B*100_000+U*10_000+B*1000+L*100+I*10+K
                                    if LIK*LIK==BUBLIK:
                                        print (LIK,'*',LIK,'=',BUBLIK)
```

Почти любую задачу в программировании можно решить разными способами, вот и я здесь проверку переменных на несовпадение значений сделал другим способом. В остальном всё как обычно: циклы по всем переменным и проверка результата умножения. Вот и ответ:

376 * 376 = 141376

Вторая задача даже чуть-чуть проще из-за условия "наибольшая цифра в слове "СИЛЕН" равна 5"; но и немного сложнее из-за большего числа переменных (их здесь 7 штук):

```
for R in range(1,10):
    for E in range(1,6):
        for Sh in range(10):
            for I in range(6):
                for S in range(1,6):
                    for L in range(6):
                        for N in range(6):
                            if R==E or R==Sh or R==I or R==S or R==L or R==N:
                                continue
                            if E==Sh or E==I or E==S or E==L or E==N:
                                continue
                            if Sh==I or Sh==S or Sh==L or Sh==N:
                                continue
                            if I==S or I==L or I==N:
                                continue
                            if S==L or S==N:
                                continue
                            if L==N:
                                continue
                            REShI=R*1000+E*100+Sh*10+I
                            ESLI=E*1000+S*100+L*10+I
                            SILEN=S*10_000+I*1_000+L*100+E*10+N
                            if (REShI+ESLI)==SILEN:
                                print (REShI,"+",ESLI,'=',SILEN)
```

Я ещё немного оптимизировал программу: переменные R, E, S не могут принимать значения 0 (ведь числа REShI, ESLI, SILEN не могут начинаться с 0). На экране мгновенно появляется решение:

9382 + 3152 = 12534

Упражнение 7.8

Сначала приведём типичную программу начинающего программиста:

```
for i in range(101,1213270,171):
    if i == 154856:
        print('Список содержит число 154856')
else:
    print('Список не содержит число 154856')
print ('Hasta la vista')
```

В этой программе одна существенная и две ... э... стилевые, что ли, ошибки.

Нажимаем F5 и изумлённо читаем на экране:

```
Список содержит число 154856
Список не содержит число 154856
Hasta la vista
```

Что-то пошло не так... Так в списке есть или нет это число?

После того как программа нашла-таки 154856, она печатает сообщение об этом радостном событии и ... продолжает крутить список из `range` дальше, что уже нехорошо: дальше-то идут числа больше 154856! "Программа выполняет то, что вы ей приказали делать, а не то, чтобы вы хотели, чтобы она делала"[10].

А потом цикл `for` заканчивается, и, так как оператора `break` программа не встретила, она выполняет оператор `else`, запутывая программиста противоречивыми выводами. Отсюда вывод: после первого оператора `print` должен был быть наш любимый оператор `break`! Исправляемся:

```
for i in range (101,1213270,171):
    if i==154856:
            print('Список содержит число 154856')
            break
```

10 Шутливый (но совершенно правильный!) третий закон Грида из вышеупомянутых законов Мерфи

```
else:
     print('Список не содержит число 154856')
print ('Hasta la vista')
```

Вот теперь другое дело:

```
Список содержит число 154856
Hasta la vista
```

Всё хорошо, программа работает быстро, но... Есть нюанс! Допустим, поступила новая вводная: теперь, видите ли, надо проверить последовательность на число 1213175. Оп-па! Вот вам и первая стилистическая ошибка – вам теперь надо вручную исправить в своей программе аж 3 числа (это ещё повезло... а могло быть и 133 числа, если программа достаточно большая!). Отсюда вывод: все наши 4 константы следует определить в начале программы на предмет возможных изменений. Переделываем программу, добавляя комментарии (исправляем 2-ую стилистическую ошибку):

```
MAX=1213270 # Верхний предел
bgn=101       # Нижний предел
Step=171      # Шаг
SEEK=1213175  # Искомое число
for i in range(bgn,MAX,Step):
    if i==SEEK:  # число найдено?
        print('Список содержит число',SEEK)# Найдено!
        break      # Выходим из цикла
else:              # Число не нашлось
    print('Список не содержит число',SEEK)
print ('Hasta la vista')  # Конец программы
```

Жмём F5:

```
Список содержит число 1213175
Hasta la vista
```

Вот теперь порядок!

Небольшое уточнение: если по запарке вдруг поступила

вводная SEEK=1213270 и вы, не моргнув глазом, внесли исправление и нажали F5, то вы получите ответ:

Список не содержит число 1213270
Hasta la vista

Но это *неверный* ответ! Переменная SEEK равна верхнему пределу и входит в последовательность! В чём же дело? Да в том, что значение *конец* (у нас это переменная MAX) НЕ ВКЛЮЧАЕТСЯ в последовательность, создаваемую функцией *range*. В связи с этим оператор *for* надо переписать так:

```
for I in range(bgn, MAX+1,Step):
```

И наконец-то увидеть на экране верный ответ:

Список содержит число 1213270
Hasta la vista

Да, а с какого перепугу Терминатор в конце второго фильма вдруг заговорил по-испански (Hasta la vista – это "до встречи" по-испански)? А с того, что в Америке сейчас больше 60 миллионов человек говорит по-испански, и это количество стремительно растёт. Так что создатели фильма просто уловили тренд – в будущем большинство американцев будет говорить по-испански (ну, или по-китайски). "...Ç'est la vie...[11]"

Упражнение 7.9

Да уж... И как прикажете решать эту задачу? Перебором? Брать все числа из отрезка [200 000 000; 400 000 000] делить их сначала на 2, потом на 3 и каждый раз проверять, чтобы остаток был 0, а итоговое частное – 1? Можно и так, только такое решение займёт очень много времени – и Питону, "великому и ужасному[12]" надо проделать 200 000 000 таких операций, а это не каждый волшебник способен проделать быстро...

11 Такова жизнь (французский афоризм). Произносится «Се ля ви»

12 Помните чудесную сказку А. Волкова «Волшебник Изумрудного города»? «Гудвин, великий и ужасный» именно оттуда

Давайте подойдем к решению с другой стороны: попробуем *сконструировать* нужные числа, то есть переберём теперь числа m и n и составим числа $N = 2^m \cdot 3^n$, попадающие в отрезок [200 000 000; 400 000 000]. Это хорошо, а до каких значений будем перебирать m и n? Узнать это очень просто – у нас ведь в руках могучий Питонище – напишем микропрограмму:

```
for i in range(100):
    M=2**i
    if M>400_000_000:
        break
print (i)
```

Результат – 29. То есть 2^{29}> 400 000 000 и больше 29 увеличивать m и, уж тем более n, не имеет смысла. Так что программа получается очень простой:

```
for m in range(2,30,2): #переберём чётные числа
    for n in range(1,30,2): #...а здесь нечётные
       N=2**m*3**n # Соберём число
       if 200_000_000 <= N <= 400_000_000: #попали в интервал?
           print (N) #печатаем подходящее число
```

Результат:

229582512

322486272

254803968

201326592

Так что сложнейшая задача, при правильном подходе (то есть применении правильного алгоритма), решается легко и просто! Между прочим, это одна из важнейших (и сложнейших) задач программиста – создать эффективный алгоритм.

Упражнение 10.1

Будем предполагать, что у нас нормальная дверь, в которой высота больше ширины; и нормальный пользователь, который не путает ширину с высотой (возглас из Одессы: "Я вас таки умоляю – и где вы таких берёте?"). Ящик пройдёт в дверь, если его второй размер меньше высоты двери, а минимальный – меньше ширины. Тогда получим программу:

```
# Вводим размеры ящика
a=int(input('введите 1-ый размер:'))
b=int(input('введите 2-ой размер:'))
c=int(input('введите 3-ий размер:'))
# Вводим размеры двери
H=int(input('высота двери:'))
M=int(input('ширина двери:'))
# Расчет минимального и максимального размеров ящика
Max=max(a,b,c)
Min=min(a,b,c)
Midi=a+b+c-Max-Min # а здесь вычисляем третий (средний) размер ящика
# пройдёт, если высота больше среднего размера,
# а ширина - минимального
if H>Midi and M>Min:
     print ('  Пройдёт!')
else:
     print ('           Не пройдёт!!!')
```

Упражнение 10.2

Основная проблема, возникающая при выполнении этого упражнения – как нам наращивать точность при округлении. Не писать же, в самом деле, что-то вроде:

```
x=2**0.5
print (round(x,1))
print (round(x,2))
print (round(x,3))
```

```
...
print (round(x,8))
```

Это просто некрасиво, в конце концов! "Некрасивые самолёты не летают [13]", а некрасивые программы, очевидно, плохо работают!

А с чего вы взяли, что второй параметр N функции round(x,N) должен быть цифрой? Смелее, не бойтесь, ставьте туда переменную – Питон не кусается [14]! Программка получилась красивой и проще некуда:

```
R=2**0.5
for i in range(8):
    print (round(R,i))
```

На экране:

1.0
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136

При извлечении корня Питон считает с точностью 16 цифр после запятой, так что можно было повторить цикл 16 раз.

Упражнение 10.3

```
def Sqr(d): # функция расчёта площади круга
     return (3.14*d**2)/4

Ro=8.5 # Удельный вес латуни
h=1 # Толщина шайбы в мм
Dmax=float(input( "Наружный диаметр, мм:"))
```

13 Широко известная "крылатая фраза" авиаконструктора А. Н. Туполева (хотя, иногда её приписывают А. С. Яковлеву и даже О. К. Антонову)

14 Питон (который змея) тоже не кусается – он глотает добычу целиком. Но человека вряд ли сможет проглотить – максимум шакала. После чего может не есть полтора (!) года

```
Dmin=float(input( "Внутренний диаметр, мм:"))
S_ring=(Sqr(Dmax)-Sqr(Dmin))# Площадь кольца в кв. мм
S_ring=S_ring/100 # Площадь кольца в кв. см
Mass=S_ring*(h*0.1)*Ro #Вес кольца
print (" вес шайбы", Mass, "г")
```

Ранее я уже рекомендовал все константы вводить в виде переменных в начале программы. Представьте, что удельный вес у нас используется в 5 местах программы, а теперь нам надо его изменить. Исправлять в 5 местах программы? Упаси вас Эйлер [15] от этого!.. Можно ведь где-то и забыть исправить, а потом мучительно искать, почему программа выдаёт неверные результаты. А так – исправили значение Ro, и всё в порядке. Именно поэтому удельный вес и толщина шайбы введена через переменные. В формуле площади круга я специально поставил скобки – везде, где вы сомневаетесь в порядке расчётов, ставьте скобки! Вот я сомневаюсь – вдруг Питон возведёт d не в квадрат, а в степень 2/4? Поставил скобки – и можно спать спокойно – сомнения исчезли!

Упражнение 10.3.1

Обычно для проверки дают задачу, которую легко проверить в уме. Например, введём оба диаметра как нули. Ответ – вес шайбы 0 граммов. Уже неплохо! А теперь введём наружный диаметр – 10 мм, а внутренний 0. Ответ: вес шайбы 0.667250000000001 **г. Прикинем: вес латунного кубика со стороной 10 мм составляет 8,5 грамм, а здесь высота в 10 раз меньше, да ещё перед нами круг, а не квадрат. Верно программа считает!**

Упражнение 10.4

Как нам известно, формула для расчёта объёма пирамиды выглядит так:

15 Леонард Эйлер — швейцарский, прусский и российский (!) математик и механик. Считается одним из величайших математиков в истории

```
V=S*h/3
```

где h – высота пирамиды, а S – площадь основания.

Программа получилась очень простой:

```
V=lambda S,h: S*h/3
VPyr0=V(140*140,49)
VPyr1=V(22*22,7)
print ('Объём пирамиды Луны примерно',int(round(VPyr0-VPyr1,
-3)), 'куб.м')
```

и выдаёт вот такой результат:

```
Объём пирамиды Луны примерно 319000 куб.м
```

Заметили, что я округлил результат до *тысяч* функцией `round()`? Нет? Вам интереснее, откуда в Египте появилась вдруг пирамида Луны? Нет, эта пирамида находится не в Египте. В Мексике, Гватемале, Белизе также имеется несколько десятков пирамид и наша находится в Мексике, в Теотиуакане. А вот кто её построил – непонятно: цивилизация Теотиуакан, которая создала эту пирамиду (и множество других) неожиданно исчезла, скорее всего из-за извержения вулкана. Ацтеки пришли уже позже...

Упражнение 10.5

Вот здесь без пользовательской функции программа будет очень неуклюжей, а так получилось простой и элегантной!

```
def MaxDiv(X): #расчет максимального делителя
    I=1 #первоначальный делитель
    for i in range(2,X-1):
        if X%i==0:#найден другой, больший, делитель!

            I=i
    return I #вернём максимальный делитель

print (MaxDiv(124)-MaxDiv(303))
```

На экране:

-39

Упражнение 10.6

Программа, может быть и длинновата, зато считает быстро и точно:

```
def Fun(x):
    return 1.24*x**5+0.12689*x**2-100.45*x-36.235
x0=3
x1=4 # начальные точки
f0=Fun(x0)
f1=Fun(x1)# начальные значения функции
delta=x1-x0 # размер отрезка
while delta>0.02: # достигнута точность?
    x01=(x1+x0)/2 # делим отрезок пополам
    f01=Fun(x01)# среднее значение функции
    if f0*f01<0: # слева и в середине разные значения?
        x1=x01 # правое значение берём из середины
        f1=f01
    else:   # в середине и справа разные значения
        x0=x01 # левое значение берём из середины
        f0=f01
    delta=x1-x0 # новый размер отрезка
print ('Корень=', (x1+x0)/2) # печатаем корень
```

А вот и ответ:

Корень= 3.0859375

Понятно, что в функцию `Fun()` вы можете вставить любую другую функцию, но тогда вам придётся вычислить другие начальные значения отрезка.

Упражнение 11.1

Задача проще пареной репы. Вот такой код я мигом набросал:

```
import math
while True: #бесконечный цикл
    a=input('введите основание логарифма ')
    if a=='q' or a=='Q':
        break #выход из программы
    b=input('число, от которого надо взять логарифм ')
    X=math.log(float(b),float(a)) # вычисляем значение логарифма
    print ('Ответ:',X)

print('\n     до новых логарифмов!!')
```

Проверяем:

```
введите основание логарифма 2
число, от которого надо взять логарифм 8
Ответ: 3.0
введите основание логарифма 2
число, от которого надо взять логарифм 0.25
Ответ: -2.0
введите основание логарифма q
     до новых логарифмов!!
```

Всё прекрасно работает, да вот беда – меня гложет червь сомнения.

ОДЗ!!!! Мы не проверяем область допустимых значений! Вот так-то: поспешишь – логарифм насмешишь. "Docendo discimus![16]**"**

Результат: дополнительные проверки существенно усложнили программу…

```
import math
while True: #бесконечный цикл
    a=input('введите основание логарифма ')
    if a=='q' or a=='Q':
```

16 Уча, мы учимся сами. Очередная латинская пословица

```
            break #выход из программы
    else:
        a=float(a)
        if a<=0:
            print ('основание логарифма должно быть > 0')
            continue
    b=float(input('число, от которого надо взять логарифм '))
    if b<=0:
        print ('число должно быть > 0')
        continue
    X=math.log(b,a)
    print ('Ответ:',X)
print('\n     до новых логарифмов!!')
```

Вот теперь всё работает корректно:

```
введите основание логарифма 2
число, от которого надо взять логарифм -0.5
число должно быть > 0
введите основание логарифма 2
число, от которого надо взять логарифм 0.5
Ответ: -1.0
введите основание логарифма q

     до новых логарифмов!!
```

Упражнение 11.2

Ну, для начала, вы, конечно же, написали примерно вот такую программу:

```
from math import *
def Ugol(b,c,a):
```

Находим значение cosα:

```
COS = (b**2+c**2-a**2)/(2*b*c)
```

Находим угол α (в радианах):

```
ALF=acos(COS)
```

Переводим радианы в градусы (функция `degrees()`) и затем округляем результат с точностью до 1 знака после запятой (функция `round()`)

```
    return round(degrees(ALF),1)
```

Вводим значения сторон:

```
a=int(input("Введите значение 1-ой стороны: "))
b=int(input("Введите значение 2-ой стороны: "))
c=int(input("Введите значение 3-ей стороны: "))
```

Выводим значения углов:

```
print(' Напротив стороны 1 угол',Ugol(b,c,a),'градусов')
print(' Напротив стороны 2 угол',Ugol(c,a,b),'градусов')
print(' Напротив стороны 3 угол',Ugol(a,b,c),'градусов')
```

И это правильная программа! Почти... Почему?! Она же всё правильно считает!

Давайте проверять. Вот мои обещанные 3 проверочные задачи:

Пусть наш подопытный треугольник – *равносторонний*. При запросе каждый раз вводите равные значения. Ответ:

```
Напротив стороны 1 угол 60.0 градусов
Напротив стороны 2 угол 60.0 градусов
Напротив стороны 3 угол 60.0 градусов
```

Отлично! Теперь треугольник пусть побудет равнобедренным: 2 стороны по 100, третья 10. Имеем на экране:

```
Напротив стороны 1 угол 87.1 градусов
Напротив стороны 2 угол 87.1 градусов
Напротив стороны 3 угол 5.7 градусов
```

Ну? Чего же больше? "Всё хорошо, прекрасная маркиза! [17]"

Хорошо, а теперь наоборот: 2 стороны по 10, третья – 100. Запускаем... Ужас!!!

```
File "C:\Как бы диск D\Питоновы программы\ugol.py",line 4,in Ugol
ALF=acos(Alf)
ValueError: math domain error
```

Что в переводе (примерном) означает "ошибка в области математики", а если точнее – в операторе `acos()`. Не выполняется неравенство треугольника: 2 стороны по 10 – это всего 20, а третья сторона у нас 100! Таких треугольников не бывает – 2 стороны по длине меньше третьей...

Увы, мы не сделали "защиту от дурака [18]", а это очень важный момент в создании программ, которые общаются с человеком. "Errare humanum est [19]", и это надо учитывать при написании программ. Значит, нам надо, во-первых, убедиться, что вводимые данные есть числа, и во-вторых, проверять введённые числа на соответствие неравенству треугольника (a<b+c, b<a+c, c<a+b). И если вторая проверка осуществляется легко и просто:

```
if (a+b)<=c or (a+c)<=b or(c+b)<=a:
    print ("        ЭТО НЕ ТРЕУГОЛЬНИК!")
```

то с первой большая проблема: если пользователь вместо числа введёт любой другой символ, то Питон сильно рассердится (а с рассерженным питоном лучше не связываться, особенно если это настоящая змея):

```
a=int(input("Введите значение 1-ой стороны: "))
ValueError: invalid literal for int() with base 10: 'л'
```

Буквальный перевод: "неверный литерал для `int()` с

17 Выражение "всё хорошо, прекрасная маркиза" используется, как иллюстрация отрицания и неуклюжей попытки скрыть реальное состояние дел

18 Защита от дурака – почти официальный термин, не несущий в себе какого-то обидного смысла, а означающий защиту устройства или программы от ошибочных действий человека

19 Человеку свойственно ошибаться. Латинская поговорка

основанием 10". Как обычно, надо доперевести на русский: "введены недопустимые данные для функции `int()`". Всё верно – функция `int()` не может превратить букву в число, даже если очень постарается. Но, как говорится, "всё учтено могучим ураганом [20]" – в Питоне есть строковый метод `.isdigit()`, который сообщает, состоит ли строка из цифр. Если не из цифр – возвращается `False`.

Уф! В результате доработки, программа, к сожалению, значительно выросла:

```
from math import *
def Ugol(b,c,a):
    Alf = (b**2+c**2-a**2)/(2*b*c)
    ALF=acos(Alf)
    return round(degrees(ALF),1)
```

Для начала запускаем бесконечный цикл, из которого выйдем оператором `break`, когда пользователь наконец выполнит все наши требования:

```
while True:
    a=input("Введите значение 1-ой стороны: ")
```

Нам нужно, чтобы пользователь ввёл только число, причём целое.

```
    if not (a.isdigit()):
```

Если введены любые другие знаки, кроме цифр...

```
        print('   Вводите только числа, пожалуйста')
```

... то просим вводить только числа...

```
        continue
```

... и возвращаемся вновь к началу нашего бесконечного цикла:

20 Фраза из бессмертного романа Ильфа и Петрова "Золотой телёнок". Здесь имеется в виду тонкий расчёт, хорошо продуманная комбинация, за которой стоит сильный ум и влиятельная организация

```
b=input("Введите значение 2-ой стороны: ")
if not (b.isdigit()):
   print('   Вводите только числа, пожалуйста')
   continue
c=input("Введите значение 3-ей стороны: ")
if not (c.isdigit()) :
   print('   Вводите только числа, пожалуйста')
   continue
```

Введённые данные выдержали проверку и оказались целыми числами строкового типа.

Превратим их в просто целые:

```
a=int(a); b=int(b); c=int(c)
if (a+b)<=c or (a+c)<=b or(c+b)<=a:
```

Если введённые числа не образуют треугольник, то печатаем соответствующее сообщение и вновь возвращаемся в начало цикла `while`...

```
    print ("         ЭТО НЕ ТРЕУГОЛЬНИК!")
    continue
```

Введены корректные данные, можно смело рассчитать углы:

```
else:
   print(' Напротив стороны 1 угол',Ugol(b,c,a),'градусов')
   print(' Напротив стороны 2 угол',Ugol(c,a,b),'градусов')
   print(' Напротив стороны 3 угол',Ugol(a,b,c),'градусов')
```

Результаты выведены на печать, прерываем цикл и завершаем программу:

```
    break
```

Всё почти прекрасно, но меня дико раздражают трижды повторенные операторы ввода с проверкой вводимых значений методом `.isdigit()` – такие повторы просто требуют превратить их в функцию! Кроме того, в данном

варианте программы, если я ошибся при вводе 3-ей стороны, мне приходится заново вводить все 3 числа. Немного переделаем программу:

```
# программа по теореме косинусов вычисляет углы треугольника
#                по трём введённым сторонам
from math import *
def VVOD (STR):
   while True: #цикл работает, пока не поступит целое число
# строковая переменная STR добавляет значение стороны
        a=input("Введите значение "+STR) #ждём число
        if not (a.isdigit()): #введено целое число?
            print('   Вводите только целые числа, пожалуйста')
        else:
            return int(a) #Вернём целое число
def Ugol(b,c,a):
    Alf = (b**2+c**2-a**2)/(2*b*c) #вычисляем cos угла по
теореме косинусов
    ALF=acos(Alf)   #угол в радианах
    return round(degrees(ALF),1) #переведём в градусы

while True: #запускаем бесконечный цикл
   a=VVOD('1-ой стороны:')
   b=VVOD("2-ой стороны:")
   c=VVOD('3-ей стороны:')# ждём корректные значения длин сторон
   if (a+b)<=c or (a+c)<=b or(c+b)<=a: #проверяем длины сторон
на неравенстве треугольника
        print ("         ЭТО НЕ ТРЕУГОЛЬНИК!")
        continue
   else: # выводим результаты
        print(' Напротив стороны 1 угол',Ugol(b,c,a),'градусов')
        print(' Напротив стороны 2 угол',Ugol(c,a,b),'градусов')
        print(' Напротив стороны 3 угол',Ugol(a,b,c),'градусов')
        break
```

Программа стала короче на 4 строки!

У этой программы имеется небольшой недостаток – вводимые данные должны быть целыми числами. А что, если в моём треугольнике стороны 11,2 см, 11,3 см, и 30,3 см? Очень просто

– пересчитайте данные в миллиметры. Пока так, а в главе с многообещающим названием "Что-то пошло не так" я всё же расскажу, как бороться с такими проблемами.

Упражнение 11.3

Ну, например, можно вот так:

```
T=True    # Это логическое выражение для оператора while
from random import *    #грузим модуль random
while T:   #Бросаем кубик, пока T=True
    a=input("Нажмите любую клавишу. Для окончания игры нажмите q
")
#q в русской раскладке клавиатуры это й
    if a!='q' and a!='й':     # нажатая клавиша не q и не й
        print (randint(1,6)) # печатаем выпавшее значение
    else:      # нажата клавиша q (или й, если включена русская
раскладка)
         print ("+"*10,"игра окончена", "+"*10)
         T=False      # а вот теперь присваиваем T значение False,

                  # после проверки в заголовке while цикл прервётся
```

Упражнение 11.4

Главное – не забыть импортировать модуль `random`:

```
import random
Sum=0 #сумматор
N=100000 #количество повторов
for k in range(N):
    Ran=random.random() #случайное число в интервале (0;1)
    Sum=Sum+Ran #суммируем случайные числа
It=Sum/N # среднее арифметическое

print (It) # печатаем результат
```

Результат, конечно, близок к 0.5 (то есть генератор случайных чисел работает неплохо):

0.49914320356186886
0.4992609692492002
0.5010038287208676
0.5001049631504257

Это результат 4 запусков программы.

Упражнение 12.1

Самое здесь сложное – это как организовать знакопеременность ряда? Подсказываю: через возведение в степень $(-1)^N$. Если N – чётное, результат +1, если нечётное, то получим -1. Тогда функция для вычисления ряда будет выглядеть вот так:

```
# программа считает сумму ряда 1-(1/3)+(1/5)-(1/7)+…

# N -кол-во повторов; m – значение знаменателя дроби
def Row(Sum, N, m):
    Sum=Sum+(1/m)*(-1)**N # значение суммы
    if N<=0:
        return Sum # прекращаем вычисления
    return Row(Sum,N-1,m+2) #вызываем функцию, сменив параметры
print ('Сумма ряда 1-(1/3)+(1/5)-(1/7)+… =',Row(1,991,3))
```

Запускаем, читаем:

```
Сумма ряда 1-(1/3)+(1/5)-(1/7)+… = 0.7856499256699722
```

А как проверить, правильно ли посчитана сумма? Очень просто. Дело в том, что точное значение суммы этого ряда имеет совершенно неожиданное значение $\frac{\pi}{4}$= 0.7853981633974483 (откуда здесь берётся число π ?!). Ну что ж, совпадение неплохое!

Упражнение 12.2

Примерно вот так:

```
def SummDig(n): #функция суммирует цифры заданного числа
    if n<10: # последняя цифра
        return n #...её и вернём
    else: #ещё не последняя цифра
    #возвращаем остаток от деления (т.е. последнюю цифру)
    #... и снова вызываем функцию, отбросив последнюю цифру
        return (n%10 + SummDig(n//10))
#вызываем функцию и печатаем результат
print (SummDig(12345678912345678912345678901234567890))
```

На экране мгновенно сумма цифр этого громадного числа:

180

Упражнение 12.3

```
def Eu(N1, N2):    #функция вычисляет НОД заданных чисел
    if N1 == N2:  #останавливаем вычисления, когда числа
сравнялись
        return N2 #возвращаем результат
    if N1>N2:
        return Eu(N1-N2,N2)# из большего вычитаем меньшее и снова
                           #вызываем функцию
    else:
        return Eu(N1,N2-N1)# из большего вычитаем меньшее и снова
                           # вызываем функцию
L=282 # собственно программа
M=7101
print (Eu(L,M)) #печатаем НОД, найденный посредством
функции Eu(N1,N2)
```

На экране:

3

"Gloria victoribus! [21]"

21 "Слава победителям!" – очередная латинская пословица

Упражнение 13.1

Ну, допустим, вот так:

```
Txt= 'Наша Таня громко плачет: уронила в речку мячик'
LenTxt=len(Txt) # число символов
N_word=5 # Которое по счёту слово надо найти
cnt=N_word-1 # счётчик пробелов
i=0 # счётчик символов
while cnt: # цикл, пока не найдём cnt пробелов
    i+=1 # счётчик символов +1
    if Txt[i]==' ': # найден пробел?
        cnt-=1 # счётчик пробелов -1
        for j in range(i+1, LenTxt): # ищем второй пробел, т.е.
конец слова
            if Txt[j]==' ':
                Txt2=Txt[i+1:j]# присваиваем срез между двумя
пробелами
                break # слово найдено, выходим из цикла
print(Txt2) # печатаем Txt2
```

Всё верно: на экране

```
уронила
```

Обращаю ваше внимание, что программа получилась легко переналаживаемой: если в переменную `Txt` вбить другую цитату, например *Txt='Голубой саксонский лес Снега битого фарфор'* [22]**, а переменной `N_word` присвоить 4, нажать F5 и получить ответ:**

```
Снега
```

Упражнение 13.2

Для начала надо рассказать вам, как проводилось шифрование, не переписывать же "Утейимцйпґ$сй$хчиґц" от руки?

Пожалуйста:

22 Великолепное стихотворение И. Бродского "В горах"

```
DeShifr='' #сюда загрузим шифрованный текст
Shifr='Слона-то я и не приметил [23]' #шифруемый текст
sdvig=4 #значение сдвига
for i in range(len(Shifr)): #цикл по всей шифровке
    q=ord(Shifr[i]) #получаем код символа
    q=q+sdvig # сдвигаемся в сторону в таблице Юникод
    n=chr(q)# новый символ
    DeShifr=DeShifr+n# добавляем символ в шифровку
print(DeShifr)# печатаем шифровку
```

Текст для дешифровки я взял, разумеется, другой, в шифрованном виде он примет вид:

Хптсд1цт$ѓ$м$сй$уфмрйцмп

Для дешифровки применяется похожая программа, примерно вот такая:

```
Shifr='Утейимцйпѓ$сй$хчиѓц'#текст для дешифровки
print (Shifr)# печатаем его
while True: #бесконечный цикл, пока не удастся расшифровать
    key=int(input('число '))# Ждём предполагаемый ключ
    DeShifr='' #сюда загрузим дешифрованный текст
    for i in range(len(Shifr)): #цикл по всей шифровке
        q=ord(Shifr[i]) #получаем код символа
        q=q-key # сдвигаемся в сторону в таблице Юникод
        n=chr(q)# новый символ
        DeShifr=DeShifr+n# добавляем символ в дешифровку
    print(DeShifr)# печатаем дешифровку
    yes=input('Если удалось дешифровать - нажмите "да" ')# текст
принял нормальный вид?
    yes=yes.lower()  #переводим ответ в нижний регистр
    if (yes=='да' or yes=='lf'): #...если ответ ДА...

        break # ... то завершаем цикл
```

Прокомментирую последние 4 строки программы.

23 Это из басни И. А. Крылова "Любопытный"

Пользователь от радости, что удалось взломать шифровку, может ведь набрать "Да", а то и вовсе "LF", забыв переключить раскладку. Методом `.lower()` переводим введённый текст в нижний регистр (строчные буквы), а дальше проверяем, что введено действительно "да" или "lf", после чего оператором `break` выходим из программы.

Начинаем дешифровку с отрицательных чисел – мы ведь не знаем, в какую сторону сделан сдвиг при шифровании:

```
число -2
Хфзлкошлcs&ул&чщкsш
Если удалось дешифровать - нажмите "да"  нет
```

-2 не подходит, -3,-4,-5 – только хуже:

```
число -3
Цхимлпщмті'фм'шъліщ
```

Попробуем положительные числа:

```
число 1
Тсдизлхиођ#ри#фцзђх
```

Вводим 2, 3 – не годится, 4 – УРА! – на экране:

```
число 4
Победителя не судят[24]
Если удалось дешифровать - нажмите "да"  да
```

Вот так, достаточно легко взломали шифровку! Набираем "да" и программа останавливается.

24 Слова эти, ставшие крылатыми, приписываются Екатерине II. Молодой А. В. Суворов, отличавшийся самостоятельностью в своих действиях, был предан военному суду за штурм в 1773 году крепости Туртукай, предпринятый вопреки приказанию фельдмаршала Румянцева. Однако рассказ о самовольных действиях Суворова при взятии Туртукая и об отдаче его под суд не вполне исторически достоверен

Упражнение 13.3

Сразу не получилось? "Ну что ж! — сказал Пудик. — Всему сразу не научишься!"[25]

А скрипт примерно такой:

```
print(str(int(input("Введите целое число:"))**2)[-1])
```

Ну это просто шедевр лаконичности Питона!

Сначала через функцию `input` вводим число в виде строки, функция `int` превращает полученную строку в число, которое тут же возводим в квадрат. Как отрезать от результата последнюю цифру? Можно, конечно, классическим способом: `print((int(input("Введите целое число:"))**2)%10)`; но нам надо обязательно применить функцию `str()`,так что преобразуем полученный результат назад в строку и печатаем последний знак.

Упражнение 13.3.1

А вот теперь это элементарно:

```
print(str(int(input("Введите целое число:"))**2)[0])
```

Это не более чем пример изумительных возможностей Питона, на самом деле ТАК писать программы не рекомендуется. Лучше (и понятней для самого себя) проделать всё это по частям:

```
Int=input("Введите целое число:") # Вводим целое
Int=int(Int) #Делаем из строки число
Out=Int**2 # Возвели в квадрат
Out=str(Out) # Превратили в строку

print(Out[0]) # Вывод ПЕРВОЙ цифры
```

25 М. Горький "Воробьишко"

Упражнение 13.4

Примерно так:

```
Chehov='''Подумав немного, он положил на блины самый жирный кусок
сёмги, кильку и сардинку,
потом уж, млея и задыхаясь, свернул оба блина в трубку, с чувством
выпил рюмку водки,
крякнул, раскрыл рот...
Но тут его хватил апоплексический удар.'''
CHEH=Chehov.lower() #в нижний регистр
while True: # вечный цикл
    fraza=input("жду фразу ") # Ждун
    if fraza.isdigit(): #фраза= цифре?
        break           # да=выход из цикла
    C=CHEH.count(fraza) #метод count считает число вхождений
    print(fraza, ':', C) #выводим результат

print ("работа закончена")
```

Ну и как? Как быстро вы сосчитали, сколько раз в этом отрывке встречается фраза '*ил*' ? 4 раза? Правильно!

Между прочим, мы левой ногой решили вполне серьёзную лингвистическую задачу! Например, лингвисты сосчитали без всяких компьютеров в середине (!) XX века, что в поэме Пушкина "Полтава" всего примерно 13 000 букв, но "Ф" встречается только 3 раза. Представляете, какой адский труд? Да ещё и чреватый ошибками: с помощью небольшой добавки к приведённой выше программе я уточнил, что в "Полтаве" 39796 букв (а вот "Ф" действительно встречается только 3 раза). Мы можем переменной `Chehov` присвоить любой текст и в доли секунды произвести над ним вполне серьёзный лингвистический анализ по наличию любых фраз.

Упражнение 13.5

```
Stroka="И Вере наша Саша не реви!..."
```

```
Str=Stroka.lower() #делаем все буквы прописными, т.к. начало
фразы идёт с прописной
Str=Str.replace(' ','') # Удаляем все пробелы
Str=Str.replace('.','')
Str=Str.replace(',','')
Str=Str.replace('!','')
Str=Str.replace(':','')
Str=Str.replace('?','') # Удаляем знаки препинания .,!?:
rtS=Str[::-1] # переворачиваем строку
if Str==rtS: # перевёрнутая строка совпадает с исходной?
    print(Stroka," - Палиндром!!!") # Если да - палиндром
else:
    print('Увы... - "', Stroka, '"- не палиндром') # если
нет - не палиндром
```

Я специально назвал переменную `Str`, так как знаю, что `str` – название функции, превращающей в строку всё подряд (то есть встроенный идентификатор), а в таких случаях ещё в начале книги я рекомендовал использовать заглавные буквы.

Упражнение 13.6

Программа получилась довольно простой:

```
Jerom='Нас      было четверо –  Джордж,       Уильям Сэмюэль
Гаррис,   я и  Монморенси'
Jerom2='' #сюда загрузим обновлённый текст
sch=0
for i in range(len(Jerom)-1): #идём по тексту
    if Jerom[i]!=' ' or Jerom[i+1]!=' ': # нет 2-х пробелов
подряд -
        Jerom2=Jerom2+Jerom[i] # добавим букву

    else:
        sch+=1 #счётчик пробелов +1
Jerom2=Jerom2+Jerom[i+1] #добавим последний знак
print (Jerom2) #выводим исправленный текст
print ('Удалено',sch,'пробелов')
```

В результате имеем на экране:

Нас было четверо — Джордж, Уильям Сэмюэль Гаррис, я и Монморенси
Удалено 15 пробелов

Упражнение 14.1

Здесь всё так просто, что я даже комментировать не буду:

```
A=[11,17,10,13,66,14,16.6]
for i in A:
    if i%2==0:
        print (i, end=' ')
```

Пожалуйста, вот вам и результат:

10 66 14

Упражнение 14.2

Здесь тоже всё просто:

```
Oi=[2,12,-5,-3.9,10,6,-3,5,22,-2.5,10] # исходный список
Ai=[] # новый список
for i in range(1,len(Oi)-1): # цикл по списку
    ai=(Oi[i-1]+Oi[i]+Oi[i+1])/3 #среднее арифметическое
    Ai.append(round(ai,2)) # добавим в новый список
print (Ai)
```

На экране:

[3.0, 1.03, 0.37, 4.03, 4.33, 2.67, 8.0, 8.17, 9.83]

Упражнение 14.3

Читать с закрытыми глазами – программа секретная!

```
# программа "РАЗВЕДЧИКИ"
# формируем последовательность букв
# № позиции, где стоит "au", даёт № ящика со спецгрузом
import random
f='qwertyuioplkjhgfdsazxcvbnm' #список латинских букв
Q=[] #здесь будет список букв
L=1000 #кол-во ящиков
for i in range (L):
    x=random.choice(f)
    Q.append(x)#формируем случайный список букв
Z=''.join(Q) #преобразуем в строку
AU=('au' in Z) #есть ли "au" в списке
if AU: #если есть - печатаем № ящика
    print('ящик №',Z.index('au'))
else:
    print('сегодня отгрузки не будет...')
```

Упражнение 14.4

Примерно так:

```
Lst=list('Значит, так:, автобусом, к, Тамбову,
подъезжаем,')#текст
Symb=','#Символ к удалению
L=Lst.count(Symb)# Количество символов к удалению
for i in range(L-1):
     J=Lst.index(Symb)# находим, где стоит символ
     Lst.pop(J)# удаляем символ
Q="".join(Lst)# создаём исправленную строку
print(Q)# печатаем её
```

Как я уже неоднократно писал, программа должна быть легко перенастраиваемая. Например, если завтра надо будет убрать из другого текста букву а, то мне потребуется изменить в программе 2 первые строки: ввести другой текст и задать другое присвоение `Symb='a'`.

Упражнение 14.5

Программа выглядит примерно так:

```
import random # Импорт датчика случ. чисел
A=[] #создаём пустой список A длиной 0
M=4 # на сколько символов сдвиг
Q=10 # длина списка
for i in range (Q):
    A.append(random.randint(1,100))# заполняем A случ. числами
print(A) # печатаем список A
B=A         # создаём список B
for k in range(M): # M раз повторяем сдвиг списка вправо на 1
символ
    x=B[Q-1]      # запомним в x последний элемент
    for i in range (Q-1,-1,-1): #цикл справа налево
        B[i]=B[i-1] # копируем элементы вправо
    B[0]=x #по окончании цикла последний элемент делаем первым
print(B)
```

На экране мы увидим, например:

[49, 74, 29, 61, 6, 66, 19, 86, 39, 50]
[19, 86, 39, 50, 49, 74, 29, 61, 6, 66]

При каждом новом запуске первая строчка будет другой, ведь она создаётся случайным образом!

Упражнение 14.6

Программа, конечно, не сильно сложная:

```
from random import randint
YesOrNo=0 #есть совпадение или нет
povtor=1000 #количество повторов
Grup=25 #сколько народу в группе
```

Импортируем нужный датчик случайных чисел и выставляем

константы:

```
for N in range(povtor): #большой цикл
```

Чтобы получить результат поточнее, надо провести расчёты в разных группах:

```
    DR=[] #здесь лежат дни рождения
```

Чтобы не заморачиваться с месяцами и днями рождения, будем использовать просто номер дня с начала года, а всего в году, будем считать, 365 дней – для простоты.

```
    for i in range(Grup):
        x=randint(1,365)
        DR.append(x) #случайным способом формируем список ДР
```

Итак, получили список DR с номерами дней рождения

```
    cnt=0
```

Обнуляем малый счётчик – в нём храним количество совпадений дней рождения для данного повтора

```
    for i in range(Grup):
        for j in range(i+1,Grup):
            if DR[i]==DR[j]: #Совпали ДР?
```

Сосчитаем количество совпадающих чисел в списке DR

```
                cnt+=1 #ДА
    if cnt!=0:
        YesOrNo+=1 #есть совпадение
```

А вот в `YesOrNo` храним количество групп, в которых есть совпадающие дни рождения:

```
print('Вероятность совпадения дней рождения в группе
из',Grup,'человек -',round(YesOrNo/povtor,3))
```

После запуска мы получим совершенно неожиданный результат:

Вероятность совпадения дней рождения в группе из 25 человек - 0.566

То есть в каждой второй группе из 25 человек есть люди, родившиеся в один день! По-научному это называется "Парадокс дней рождения" – мы никак не ожидаем, что в столь небольшой группе вероятность совпадения дней рождения (число и месяц) хотя бы у двух людей превышает 50 %. Для 57 и более человек вероятность такого совпадения превышает 99 % . Запустите программу при `Grup=57` и компьютер вам выдаст:

Вероятность совпадения дней рождения в группе из 57 человек - 0.992

Ну а 100 % она достигает, согласно здравому смыслу, только тогда, когда в группе не менее 367 человек (тут уж по любому у 2-х человек день рождения совпадёт – с учётом високосных лет). Программа, кстати, считает довольно точно – математически рассчитанная вероятность совпадения дней рождения в группе из 25 человек составляет 0,569[26].

Упражнение 14.7

Обдумаем алгоритм решения задачи. Сначала надо импортировать из модуля `random` метод .shuffle(), который умеет случайным образом перемешивать (то есть тасовать!) элементы списка. Определимся с обозначением масти карт: обозначим "черви" буквой "ч", "бубны" – буквой "б", "кресты" – "к", "пики" – "п". Достоинства карт обозначим большими буквами: "туз" буквой "Т", "король" – "К", "дама" – "Д", "валет" – "В". Десятки, разумеется, обозначим "10", девятки – "9" и так далее. Так что "9к" это 9 кресты, а "Дп" – пиковая дама.

26 Это при условии, что дни рождения в году распределены равномерно, а это не совсем верно. Например, в США самый распространённый день рождения – 9 сентября, а наименее распространённый – 25 декабря. Первое встречается примерно в два раза чаще, чем второе(!)

Полную колоду разобьём на 4 подколоды – исключительно для удобства. Из перетасованной методом .shuffle() колоды карты игрокам будем выдавать с помощью метода .pop() – он удаляет и выдаёт последний элемент списка – если не указать индекс. Из оставшейся колоды показываем нижнюю карту – с номером 0 – она и определяет козыри. Далее из комментариев, надеюсь, всё более-менее понятно:

```
from random import shuffle
K_cherv=['Тч','Кч','Дч','Вч','10ч','9ч','8ч','7ч','6ч']
K_bub=['Тб','Кб','Дб','Вб','10б','9б','8б','7б','6б']
K_krest=['Тк','Кк','Дк','Вк','10к','9к','8к','7к','6к']
K_pik=['Тп','Кп','Дп','Вп','10п','9п','8п','7п','6п']
KOLODA=K_cherv+K_bub+K_krest+K_pik #собираем полную колоду
Igr1=[];Igr2=[]; Igr3=[]
shuffle(KOLODA) # перетасуем колоду
for i in range(6):
    x=KOLODA.pop() # выдаём карту и сразу её убираем из колоды
    Igr1.append(x) # добавляем эту карту игроку 1
    x=KOLODA.pop() # выдаём карту и сразу её убираем из колоды
    Igr2.append(x) # добавляем эту карту игроку 2
    x=KOLODA.pop() # выдаём карту и сразу её убираем из колоды
    Igr3.append(x) # добавляем эту карту игроку 3
print('карты игрока 1',Igr1)
print('карты игрока 2    ',Igr2)   # печатаем карты игроков
print('карты игрока 3        ',Igr3)
karta=KOLODA[0] # вынимаем нижнюю карту из колоды
print ('    карта устанавливает козыри:',karta) # и печатаем её
 #                определяем козырь
if karta[1]=='п':
    koz='ПИКИ '
elif karta[1]=='к':
    koz='КРЕСТИ '
elif karta[1]=='б':
    koz='БУБИ '
else:
    koz='ЧЕРВИ '
print (koz,'козыри у нас') # печатаем козырь
print("в колоде остались карты:",KOLODA) # и остатки колоды
```

Результат работы программы каждый раз новый, но примерно такой:

карты игрока 1 ['Тб', 'Тч', 'Тк', '10к', '7б', '9ч']
карты игрока 2 ['Дб', 'Вб', '7к', 'Кп', 'Кк', 'Кб']
карты игрока 3 ['Тп', '9б', '8п', '6б', '8б', '9п']
карта устанавливает козыри: 6п
ПИКИ козыри у нас
в колоде остались карты: ['6п', '9к', '7п', 'Вп', '10ч', '8к', '10б', 'Дп', 'Дк', 'Вч', '7ч', '8ч', 'Дч', 'Кч', '10п', 'Вк', '6ч', '6к']

Упражнение 14.8

Просим у пользователя день, месяц и год рождения, а также какой *день со дня рождения* **желаете отметить:**

```
DayOfBorn=int(input('день вашего рождения '))
MonthOfBorn=int(input('месяц, когда вы родились (номер) '))
YearOfBorn=int(input('год рождения полностью '))
Days=int(input('Какой ДЕНЬ рождения желаете отметить: '))
```

Конечно, можно попросить пользователя вводить свой месяц рождения в привычной форме, но здесь я применил своеобразную "защиту от дурака": при вводе месяца в текстовой форме можно запросто ошибиться: "сегтябрь", "сетябрь"... При вводе числом вероятность ошибки меньше.

А теперь давайте придумаем алгоритм расчёта требуемой даты. Сначала рассчитаем целое число лет, прошедших за введённое пользователем число дней. Для этого поделим количество дней `Days` на 365.25 (это усреднённое количество дней в году) и добавим год рождения.

```
days=Days
God=int(days/365.25)+ YearOfBorn
```

Найдём теперь остаток от деления на 365,25:

```
days=days%365.25
```

Деление этого остатка на 30,5 (среднее количество дней в месяце) даст нам число прошедших месяцев:

```
Mes=int(days/30.5)+ MonthOfBorn
```

А вот после сложения полученного числа с месяцем рождения нас, быть может, ожидает неожиданность – суммарное число месяцев может оказаться больше 12! Ну, здесь всё понятно – уменьшаем число месяцев на 12, а число лет увеличим на 1:

```
if Mes>12:
    Mes=Mes-12
    God+=1
```

Остаток от деления на 30,5 даст нам число дней, которые надо добавить к дате рождения:

```
days=int(days%30.5) + DayOfBorn
```

Здесь аналогичная проблема – результат может превысить число дней в месяце:

```
if days>30:
    days-=30
    Mes+=1
```

И здесь результат может опять-таки превысить число месяцев:

```
if Mes>12:
    Mes=Mes-12
    God+=1
```

Мы получили номер месяца, когда мы будем праздновать свой миллионный день рождения, а для пущей красоты нам надо вывести номер месяца привычным словом: "октября", "мая"... Ну это просто – надо создать список месяцев:

```
Moons=['января','февраля','марта','апреля','мая','июня','
июля','августа','сентября','октября','ноября','декабря']
print ('Ваш',Days,'день рождения отмечайте',days, Moons
[Mes-1],God,'года')
```

Запускаем программу и...

Ваш 1000000 день рождения отмечайте 11 июля 4698 года

Вот это да! Я подозревал, что до миллионного дня рождения дожить довольно проблематично, но что до такой степени... Умерим аппетиты, рассчитаем 30 000 день рождения:

Ваш 30000 день рождения отмечайте 25 октября 2042 года

Ну вот это уже в пределах разумного. Посчитайте даты для себя, например, созовите друзей на свой внеочередной, 5555 день рождения! Или 22222. Или 12345!

Упражнение 15.1

```
Dem=tuple('''Никогда, никогда ни о чем не жалейте —
Ни потерянных дней, ни сгоревшей любви.
Пусть другой гениально играет на флейте,
Но ещё гениальнее слушали вы.''')
start=0 #начало поиска
for m in range (Dem.count('д')): #цикл по всем буквам
    i=Dem.index('д',start) #адрес следующей буквы
    print(i, end=' ') #печать адреса
    start=i+1 #продолжим поиск со следующего адреса
```

А вот и результат:

5 14 53 85

Упражнение 15.2

Программа несложная, результат в одном месте неожиданный.

```
##функция возвращает в виде кортежа:
##     - аргумент, умноженный на 2
##     - квадрат аргумента
##     - квадратн. корень из аргумента, если возможно
def multi(x):
    X=x*2
    X2=x**2
    if x<0:
        X05='нельзя извлечь корень из числа  '+str(x)
    else:
        X05=round(x**0.5,4)# округлим результат до 4 знаков
    return X,X2,X05
#  собственно программа
inT = [2,6,-2,8,2.3]
for i in inT: #берём числа из списка
    outT=multi(i) #получаем кортеж с 3 результатами
    print (outT)
```

Результат:

```
(4, 4, 1.4142)
(12, 36, 2.4495)
(-4, 4, 'нельзя извлечь корень из числа  -2')
(16, 64, 2.8284)
(4.6, 5.289999999999999, 1.5166)
```

Я вроде бы всё предусмотрел: и невозможность извлечь корень из отрицательного числа, и округлил значения квадратных корней, и, тем не менее, в последней строчке появилось значение:

```
5.289999999999999
```

Это квадрат числа 2.3?! Да я на бумажке, в столбик, точнее сосчитаю: $2,3^2=5,29$!

Вы совершенно правы. У меня всё время не было повода рассказать про эту особенность Питона: при проведении *вычислений с плавающей запятой* иногда появляются неточности в последнем, 16-ом знаке после запятой. Это надо учитывать, особенно при проведении операции сравнения. Попробуйте сравнить:

```
>>>2.3**2==5.29
```

И вы получите результат `False`!

`False`! То есть "ложь", Карл[27]! Похоже, с сегодняшнего дня отменили правила арифметики, а я ничего не знаю?! И что теперь делать? Как что делать? Округлять полученные значения хотя бы до 13-14 знаков после запятой. Я, честно говоря, даже не смог найти пример, где требуется такая высокая точность – 13-14 знаков! Если вам вдруг всё-таки потребовалось провести вычисления с сумасшедшей точностью, то в Питоне есть модуль, считающий абсолютно точно, и имя ему `Decimal`. Подробнее про этот модуль я расскажу, возможно, в какой-нибудь следующей книге, а пока – округляйте, а то иначе:

```
>>>print (0.5-0.2-0.2-0.1)
-2.7755575615628914e-17
```

То есть результат опять не равен 0!

А вот так уже лучше:

```
>>>x=round((0.5-0.2-0.2-0.1),13)
>>>x==0
True
```

27 Интернет-мем "Карл" произошёл из сериала "Ходячие мертвецы" (какое милое название!). Там отец что-то объяснял эмоционально своему сыну, а в конце сказал "Карл!" и Интернет-пользователи подхватили это и сделали мемом. "Карл" употребляется, когда хотят усилить фразу, привлечь внимание к смыслу сказанного

Упражнение 15.3

Так как задача довольно простая, то первым делом напишем вот такую простенькую программу:

```
a=(55,31,33,-40,62,-27,58,-2,5,9,-5,33,10)
S3=13 # заданное значение суммы
lena=len(a)
for i in range(lena):
    for j in range(i+1, lena):
        for k in range(j+1,lena):
            if a[i]+a[j]+a[k]==S3:
                neWa=(a[i],a[j],a[k])
                print (neWa)
                break
else:
    print('no')
```

Три цикла один в другом, если тройка нашлась – быстренько составляем новый кортеж `neWa` и командой `break` выходим из цикла. Не нашлась тройка – тогда в конце цикла оператором `else` (в цикле он применяется, если ни разу не сработала команда `break`) печатаем "no". Запускаем, и...

```
(55, -40, -2)
(31, -27, 9)
(-40, 58, -5)
(-2, 5, 10)
подходящих троек нет
```

...понимаем, что задача сложнее, чем казалось... Оказывается, в этом кортеже спрятано аж 4 (!) подходящих тройки, команда `break` была выполнена только в самом внутреннем цикле, а оператор `else` принадлежит самому первому, внешнему циклу, поэтому Питон и напечатал "подходящих троек нет", чем несказанно удивил меня.

Модернизируем программу. Информацию о том, что тройка

найдена и необходимо прервать цикл, будем передавать во внешние циклы посредством переменной `ind` ("индикатор"), и тогда оператор `else` сработает корректно. Программа написана:

```
a=(55,31,33,-40,62,-27,58,-2,5,9,-5,33,10)
S3=13 # заданное значение суммы
ind=0
lena=len(a)
for i in range(lena):
    for j in range(i+1,lena):
        for k in range(j+1, lena):
            if a[i]+a[j]+a[k]==S3:
                neWa=(a[i],a[j],a[k])
                print ('найдена тройка',neWa)
                ind=1
                break
        if ind==1:
            break
    if ind==1:
        break
else:
    print('подходящих троек нет')
```

и успешно работает. Например, если S3=13 на экране вывод:

найдена тройка (55, -40, -2)

а если S3=21 то получим:

подходящих троек нет

Тем не менее у меня остаётся чувство неудовлетворённости. Программа явно составлена как-то неэффективно: а что, если надо будет найти сумму не трёх, а тридцати трёх чисел – так и будем долго и скучно выкарабкиваться во внешний цикл? Знаете, чем отличается мудрый от умного?

"Умный с блеском выпутается из любых очень сложных ситуаций, в которых мудрый никогда не окажется[28]".

Вот и давайте поступим мудро! Коли нам надо резко выйти из целой кучи вложенных друг в друга циклов, надо вместо оператора `break` применить оператор `return` – он заставляет программу-функцию сразу прерваться в любом месте и выйти из неё!

```
def SEEK(a, S3):
    lena=len(a)
    for i in range(lena):
        for j in range(i+1,lena):
            for k in range(j+1, lena):
                if a[i]+a[j]+a[k]==S3:
                    neWa=(a[i],a[j],a[k])
                    print ('найдена тройка',neWa)
                    return
    print('подходящих троек нет')

a=(55,31,33,-40,62,-27,58,-2,5,9,-5,33,10)
S3=13
SEEK(a, S3)
```

Программа стала короче, а значит и понятней, и работает как надо. При S3=13 ответ:

найдена тройка (55, -40, -2)

при S3=21:

подходящих троек нет

Уф! Наконец-то программа получилась красивой и работающей. Да, согласен: иногда писать программы надо,

28 Кто автор этого блестящего афоризма – непонятно. Интернет путается в показаниях, но скорее всего – немецкий писатель Жан Поль. Почему у немца в XVIII веке фамилия Поль? Потому, что это псевдоним, естественно. Настоящая фамилия – И. П. Ф. Рихтер

"хорошенько накушавшись гороха"[29], но оно того стоит: какое наслаждение, когда программа наконец-то заработает!

Упражнение 15.4

Не сильно просто, но и не сильно сложно:

```
# список букв и символов для перекодировки
RUS=tuple("АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЪЭЮЯ 0123456789-,
.;:()/%!?абвгдеёжзийклмнопрстуфхцчшщьыъэюя")
KEY=1,5,-2,8,11,-4 # ключ
Stroke = 'Господствует ещё смешенье языков: Французского с
нижегородским?' # Донесение
L=len(Stroke); K=len(KEY); R=len (RUS) #Длины всех трёх массивов
Shfr=[] # Список с зашифрованной строкой
j=0 # Переменная-указатель для для кортежа KEY
for I in range(L): # Бежим по донесению
    Ltr=Stroke[I] # Последовательно выбираем буквы из донесения
    x=RUS.index(Ltr) # находим позицию символа
    Y=x+KEY[j] # добавляем сдвиг к позиции символа
    if Y>=R: # Если вышли за пределы кортежа RUS
        Y-=R  # переходим в начало кортежа
    Shfr.append(RUS[Y])  # добавляем букву
    j+=1 # увеличиваем j
    if j >=K: # если j вышел за пределы кортежа-ключа
        j=0 # переходим в начало кортежа
Res="".join(Shfr) #Превращаем список в строку
print (Res) # выводим результат
```

Да, я всё время забывал вам сказать, что операторы в Питоне можно записывать через точку с запятой – как в 4–ой строке программы; но желательно так делать только с какими-то однородными действиями – у меня это операции вычисления длин массивов.

Ну а как расшифровать полученную шифровку:

29 Цитата из повести "Белеет парус одинокий" В. Катаева. Смысл фразы: обладая огромным терпением, выдержкой

Дупчщатчаыпо0йчн-ннйцмшцё4эпЁжпж.7Ямбтфытнлубц-н0тжоп?пхмлъжйс%

Да очень просто – оператором:

```
Y=x+KEY[j]
```

мы добавляли значение сдвига и шифровали текст, а оператором:

```
Y=x-KEY[j]
```

мы его дешифруем. Вставьте в переменную `Stroke` полученную шифровку и замените значение `Y` на `Y=x-KEY[j]` – и донесение будет расшифровано.

Упражнение 16.1

Допустим, вот так.

```
Fam={"Айвазовский":13, "Петров-Водкин":11,"Репин":6,
"Куинджи":7,"Серебрякова":10}
while True :
    Im=input ("Введите фамилию") #Ждём фамилию
    if Im.isdigit(): # Введено число?
        break # завершаем работу
    Im=Im.strip() # удаляем пробелы
    Im=Im.capitalize() # Делаем 1-ую букву заглавной
    if Im in Fam:
        print ("Табельный номер", Fam[Im]) #выводим табельный №
сотрудника
         # выходим во внешний цикл
    else: # фамилия не нашлась
        print ("Нет в списке. Добавить - 'да', нет - любая другая
клавиша ")
        Y=input () #Ждем ввода
        if Y.upper()=="ДА"  : # ввели да
            Num=list(Fam.values()) # делаем список значений
```

```
            NumMax=max(Num) # вычисляем макс. значение
            Fam[Im]=NumMax+1 # добавляем в словарь новый элемент
            print (Fam)
print (" До свидания! ")
```

Упражнение 16.2

```
x = "Бойцовый кот есть самостоятельная боевая единица сама
в себе, способная справиться с любой мыслимой и немыслимой
неожиданностью"

ABC = {} # создаём пустой словарь
for i in range(1, len(x) - 1): # проход по заданной фразе
    if x[i] not in ABC:# если символа нет в словаре
        ABC[x[i]] = 1 # добавляем символ в словарь как ключ
и задаём его значение как 1

    else: # символ есть уже в словаре
        ABC[x[i]] += 1 # увеличиваем его значение на 1
print (ABC) # печатаем распределение частот
L=len(ABC) # узнаём длину словаря
Max=0 # сюда загрузим макс. число повторений
Symb=0   # сюда загрузим макс. часто встреченный символ
a=ABC.keys() # Список ключей
for j in a: #цикл по списку ключей
    if ABC[j]>Max:  # Найден более частотный символ?
        Max2=Max # Перегружаем предыдущий максимум
        Symb2=Symb  #Перегружаем предыдущий максим. символ
        Max=ABC[j] # Новое максим. число
        Symb=j # здесь храним самый частый символ
print ("Символ", Symb, "встречается", Max, "раз")
print ("А символ", Symb2, "встречается", Max2, "раз")
```

Конечно, парой строк тут не обойтись:)

Упражнение 16.3

Конечно, на первый взгляд программа простая, но... Но защита от дурака и корректный выход при нажатии цифры существенно её усложнили...

```
# Используем словарь для подсчета количества очков за собранное
слово в Эрудите

def test(): #Проверяем введённое слово

    while True:
        word = input("Введите слово: ")
        word = word.upper() #переводим все буквы в верхний
регистр
        if word.isdigit(): #если введена цифра
            return 0
        else:
            ind=1
            for j in word: #нет ли в слове латинских букв
                ind=ind*(j in points)
            if not(ind): # если была хотя бы буква на латинице,
то ind будет 0
                print ("Слово должно состоять только из русских
букв")

            else:
                return word # Слово правильное, вернём его
# Создаем словарь с соответствием букв и очков за них
points = {"А": 1, "Б": 3, "В": 1, "Г": 3, "Д": 2, "Е": 1,"Ё": 1,
 "Ж": 5, "З": 5, "И": 1, "Й":4, "К": 2, "Л": 2, "М": 2,
 "Н": 1, "О": 1, "П": 2, "Р": 1, "С": 1, "Т": 1,
 "У": 2, "Ф": 8, "Х": 5, "Ц": 5, "Ч": 5, "Ш": 8,
 "Щ": 10, "Ь": 3, "Ы": 5, "Ъ": 15, "Э": 8, "Ю": 8,"Я":3}


while True: запускаем бесконечный цикл
    word=test()
    if word!=0: # Считаем количество очков
        score = 0 #обнулим счётчик
```

```
        for ch in word: #считаем из словаря вес каждой буквы
            score = score + points[ch]
    # Выводим результат
        print("число очков", score)
    else: # функция вернула 0
        break #конец работы
print ( '-- до новых встреч! --' )
```

Проверим, как работает:

Введите слово: пастернак [30]
число очков 11

Надо же, какое длинное слово и как мало очков. Но подсчёт правильный!

Введите слово: объём

число очков 22

Верно!

Упражнение 17.1

С помощью множеств задача решается на раз-два-три:

```
Anag1=set('отсечка')
Anag2=set('сеточка')
Anag3=set('тесачок')
Anag4=set('чесотка')
if Anag1|Anag2|Anag3==Anag4:
    print('Это анаграммы')
else:
```

30 Я знаю, что в "Эрудите" нельзя использовать имена собственные. Но это слово в данном случае означает не фамилию великого, но малоизвестного советского поэта, а название также малоизвестного огородного растения типа петрушки

```
    print('Это не анаграммы')
```

На экране:

```
Это анаграммы
```

Упражнение 17.2

Первоначально задача кажется проще некуда. Делаем из панграммы множество, и если в нём 33 элемента, то перед нами панграмма!

```
PAN='Съешь же ещё этих мягких французских булок да выпей чаю'
LenPan=len(PAN)
setPAN=set(PAN)
if len(setPAN)==33:
    print ('Это панграмма. В ней',LenPan, 'букв')
else:
    print ('Это не панграмма.')
```

Запускаем и программа выносит вердикт:

Это не панграмма.

Как не панграмма?! Я же самолично всё посчитал – 33 разных буквы, а всего их – 55... В чём дело? А давайте посмотрим на состав множества `setPAN`:

```
>>>setPAN
{'л', 'ш', 'з', 'к', 'г', 'ю', 'э', 'у', 'а', 'д', 'ь', 'щ', 'е', 'м', 'о', 'т', 'б', 'п', 'ъ', 'я', 'ы', 'и', 'н', 'й', 'ё',
'ч', 'ф', 'ц', 'х', 'в', ' ', 'р', 'С', 'с', 'ж'}
```

Первое же, что бросается в глаза – в множестве две буквы "С" – заглавная и строчная, это для Питона разные буквы! И второе – пробел, естественно, считается за букву! Это мы мигом исправим – благо, есть подходящие методы.

А если строчка окажется не панграммой – добавим список

недостающих для счастья букв. Как? Да просто из эталонного множества, содержащего все 33 буквы нашего алфавита, вычтем множество, составленное из букв неудавшейся панграммы. Программа немного преобразится:

```
Etalon=set('ёйцукенгшщзхъэждлорпавыфячсмитьбю')
PAN='Съешь же ещё этих мягких французских булок да выпей чаю'
PAN=PAN.lower() #сделаем все буквы строчными
LenPAN=len(PAN)#сколько букв в исходнике
setPAN=set(PAN) #сделаем из фразы множество
setPAN.discard(' ')# удалим пробелы
if len(setPAN)==33:
    print ('Это панграмма. В ней',LenPAN, 'букв')
else:
    Netu=Etalon-setPAN # находим недостающие буквы
    print ('Это не панграмма. Нет букв',Netu)
```

Запустим... Вау! Работает!

```
Это панграмма. В ней 55 букв
```

А теперь удалим в проверяемой фразе буквы "г" и "о":

```
Это не панграмма. Нет букв {'г', 'о'}
```

Ну что ж. Хорошо. Я бы даже сказал – отлично. Поэтому изменим подопытную фразу, пусть теперь это будет: "Любя, съешь щипцы, — вздохнёт мэр, — кайф жгуч" (к сожалению, все предельно короткие панграммы какие-то идиотские – это расплата за краткость). Вставляем фразу в программу, в переменную PAN, запускаем и читаем не менее идиотский ответ:

```
Это не панграмма. Нет букв set()
```

В нашей фразе завелись знаки препинания, и длина множества setPAN превысила отметку в 33 знака! Поэтому при вычитании `Etalon-setPAN` мы получили как бы

отрицательное множество. Вы можете представить множество с отрицательным числом элементов? Вот и Питон не может. Спасибо, что хоть программа не вылетела, а выдала странный, но всё же ответ.

Значит, надо удалить из множества, содержащего знаки из подопытной фразы, всё, что не буква! Вместо строки, удаляющей пробелы, вставим в программу вот такой фрагмент:

```
for j in setPAN: #Цикл по элементам множества!
    if not (j.isalpha()):
        setPAN.discard(j)
```

Запускаем... Питону этот фрагмент не понравился:

```
for j in setPAN: #Цикл по элементам множества!
RuntimeError: Set changed size during iteration
```

Перевод: "*множество изменило размер в процессе итерации*", то есть в процессе считывания значений в заголовке цикла – мы же удаляем из множества знаки препинания, вот его длина и меняется прямо в ходе цикла. Ой, ну подумаешь! Сделаем копию множества и поработаем с ней:

```
setPAN2=setPAN#сделаем из фразы множество ещё раз
for j in setPAN2: #Цикл по значениям копии!
    if not (j.isalpha()):
        setPAN.discard(j) #всё, что не буква - удаляем из
основного множества!
```

Запускаем... И получаем от Питона такой же ответ:

```
for j in setPAN2: #Цикл по элементам множества!
RuntimeError: Set changed size during iteration
```

Внимание! *Важный момент!* **Не рекомендуется присваивать одной последовательности значение другой. На самом деле**

вы получите при этом *одну* последовательность (то есть множество, кортеж, список) с 2 именами. Вот вам пример:

```
>>>R=[2,3,4]
>>>R2=R
>>>R[1]=-5
>>>R2
[2, -5, 4]
```

То есть изменили значение в списке R, а изменился также и список R2!

Аналогичное явление произошло и у нас. Поэтому сделаем копию непосредственно из фразы:

```
setPAN2=set(PAN)
```

ну а программу запишем вот так:

```
Lno='' # здесь будет список букв, которых нет в заданной строке
Etalon=set('ёйцукенгшщзхъэждлорпавыфячсмитьбю')
PAN='Любя, съешь щипцы, — вздохнёт мэр, — кайф жгуч.'
LenPAN=len(PAN) #сколько всего букв во фразе
PAN=PAN.lower()#сделаем все буквы строчными
setPAN=set(PAN)#сделаем из фразы множество
#Избегайте прямых присваиваний типа setPAN2=setPAN. При
изменении значения
# в одной последовательности, может измениться и другая!
# Поэтому и проводится это, на первый взгляд, нелогичное
присвоение
setPAN2=set(PAN)#сделаем из фразы множество ещё раз
for j in setPAN2: #Цикл по значениям копии!
    if not (j.isalpha()):
        setPAN.discard(j) #всё, что не буква - удаляем из
основного множества!
if len(setPAN)==33:
    print ('Это панграмма. В ней',LenPAN, 'букв')
else:
```

```
    Netu=list(Etalon-setPAN)# находим недостающие буквы

# здесь мы просто организуем красивый вывод без фигурных скобок
    for j in Netu:
пробел  Lno+=j+' ' # выведем отсутствующие буквы через
    print ('Это не панграмма. Нет букв:',Lno)
```

Запускаем... Ура! Работает!

Это панграмма. В ней 47 букв

Удалим в подопытной фразе первое слово. На экране:

Это не панграмма. Нет букв: я ю б л

Упражнение 17.3

Примерно так (игру заканчиваем, как и обещал, при нажатии редкой клавиши – "Ъ"):

```
import random
abc='абвгдежзик'# список букв
Pole=set()#создаём пустое множество
for i in range(len(abc)*4):
    X=random.choice(abc)# координата X - буква
    Y=random.randint(1,len(abc))# координата Y - число
    Z=X+str(Y)#собираем пару буква-число
    Pole.add(Z)# и сохраняем пару в множестве
IN=0 # счетчики попаданий...
OUT=0 #... и промахов
while True:#
    XY=input('Стреляй ')#ждём координаты от игрока
    if XY=='ъ' or XY=='Ъ':# конец игры
        print ("У вас ",IN, " попадания и ", OUT ,"промахов!")#
        break
    elif XY in Pole:# попал!!!!
         IN+=1
         print ('      Попал!!!!')
```

```
    else: #      мимо!!!!!
        OUT+=1
        print("Мимо")
```

Упражнение 17.4

Задача оказалась посложнее, чем сначала казалось. Сначала, естественно, получилась вот такая программа:

```
import random
SKLAD=set()#создаём пустое множество
x=1000
y=1000
z=10
while len(SKLAD)<=(x*y*z*0.995): # заполняем склад на 99.5%
    X=random.randint(1,x)# координата X
    Y=random.randint(1,y)# координата Y
    Z=random.randint(1,z)# координата Z
    Q=str(X)+'-'+str(Y)+'-'+str(Z) #формируем адрес
    SKLAD.add(Q)#заполняем склад
ind=0 # если =1, значит свободная ячейка найдена
print ('готово') #склад заполнен
```

Начинаем поиск свободной ячейки поближе к входу:

```
for xi in range (1,x): #пробегаем по x
    for yi in range (1,y): #пробегаем по y
        for zi in range (1,10+1): #пробегаем по z
            Qi=str(xi)+'-'+str(yi)+'-'+str(zi) #формируем адрес
            if not((Qi)in SKLAD): #есть ли такой адрес
                ind=1
                break # выход
```

По команде `break` мы выйдем только из цикла по `Z`, а надо выйти из всех циклов сразу. Вот для этого и введена `ind` – переменная-индикатор немедленного выхода из *всех* циклов

```
        if ind==1:
            break # выход
```

Цикл по y проверил значение ind и получил команду break.

```
    if ind==1:
        break # выход
```

Цикл по x проверил значение ind и получил команду break. Вот теперь можно напечатать адрес найденной ячейки:

```
print ('свободная ячейка',Qi)# печатаем адрес свободной ячейки
```

После запуска программа пару минут формирует множество SKLAD и затем за доли секунды находит свободную ячейку – вот она:

```
свободная ячейка 1-59-7
```

Но! Вот только найденная ячейка находится как-то далековато от входа...

Ну конечно! Программа находит свободную ячейку не самым оптимальным способом...Нам желательно, чтобы сумма первых двух координат была как можно меньше, а мы двигаемся по координате x до самого конца склада! Поэтому надо подкорректировать программу поиска, постепенно наращивая сумму координат x и y.

```
for i in range (2,x+y+1): #сумма координат как минимум 2
    for yi in range (1,i+1): # координата y
        xi=i-yi          # координата y
        for zi in range (1,z+1): # координата z
            Qi=str(xi)+'-'+str(yi)+'-'+str(zi)#формируем адрес
            if not((Qi)in SKLAD): #есть ли такой адрес
                ind=1 # да, есть
                break  # выход
```

```
        if ind==1:
            break # выход
    if ind==1:
        break # выход
print ('свободная ячейка',Qi)
```

Вот это другое дело:

```
Свободная ячейка 6-4-7
```

Как видите, "Ох, тяжела ты, шапка Мономаха!"[31] (ну или работа завсклада) ...

Упражнение 18.1

Возьмём вдохновляющее высказывание Конфуция: "Путь в тысячу ли начинается с первого шага", и начнём наш путь. Сделаем из этой фразы "огромное" множество:

```
a1='Путь в тысячу ли начинается с первого шага'
SET=set(a1.split()) #сделаем из фразы множество, разбив её на
слова
```

Помните, как работает метод `.remove(x)`? Если удаляемого элемента нет в множестве, то произойдёт ошибка – этим и воспользуемся с помощью операторов `try / except`:

```
rmv=input ('что прикажете найти и обезвредить ')
try:
    SET.remove(rmv) #попытаемся удалить заданный элеменнт
    print('элемент со значением "',rmv,'" найден и удалён')
except:
    print('элемент со значением "',rmv,'" не найден')
```

31 А. С. Пушкин. Строчка из трагедии "Борис Годунов". Цитируется как жалоба на тяжёлую долю, большую ответственность руководителя

Вот и вся программа! На экране диалог:

что прикажете найти и обезвредить ли
элемент со значением " ли " найден и удалён
что прикажете найти и обезвредить км
элемент со значением " км " не найден

Упражнение 18.2

Для начала смоделируем проблемный фрагмент программы:

```
abc='кукарек'
Dobo=2
Res=abc+Dobo
print (Res)
```

и запустим его, чтобы получить название ошибки:

```
Traceback (most recent call last):
  File "C:\Как бы диск D\Питоновы программы\Программы для Трактата\
try.py", line 13, in<module>
Res=abc+Dobo
TypeError: can only concatenate str (not "int") to str
```

Пожалуйста – ошибка `TypeError`. Переделываем программу:

```
abc='кукарек'
Dobo=2 # неверные данные
try:
    Res=abc+Dobo #пытаемся сложить переменные
    print (Res)
except TypeError: # обход ошибки
    print ('обходим ошибку')
print ('работаем дальше')
```

Всё! Если `Dobo` принимает числовое значение 2, на экране:

```
обходим ошибку
работаем дальше
```

а если Dobo равно '2' – строковое значение:

```
кукарек2
работаем дальше
```

Упражнение 19.1

Можно так:

```
F=open('Medved.txt', encoding="utf-8" )# открываем файл
TXT=F.readlines() #считаем файл построчно
for i in range(len(TXT)):#печатаем построчно
    print (TXT[i],end='')
F.close() # закрываем
```

А красивее так:

```
F=open('medved.txt', encoding="utf-8" )# открываем файл
TXT=F.readlines() #считаем файл построчно
for i in TXT:#печатаем построчно
    print (i,end='')
F.close() # закрываем
```

На экране в любом случае мы увидим:

Нёс медведь, шагая к рынку,
На продажу мёду крынку,
Вдруг на Мишку — вот напасть! —
Осы вздумали напасть.

Ой… Я немного перепутал и открыл другой файл, правда, со стихотворением того же автора – Я. Козловского, и, опять же, с участием медведя :-). Так ведь даже интересней!

Обратите внимание, что название открываемого файла в разных программах написано по-разному: `Medved.txt` и `medved.txt`. Файл Питон открывает не сам, а просит операционную систему сделать это, а Windows, в отличие от Питона, нечувствительна к регистру. На самом деле открывался один и тот же файл.

Упражнение 19.2.1

Хочу подробно прокомментировать созданную программу. Сначала открываем файл, где будем хранить строку:

```
F=open('python.txt', 'w',encoding="utf-8" )# открываем файл
```

Набор букв, из которых будем создавать строку:

```
q='инпот'
```

Список, куда будем сохранять буквы:

```
FF=[]
```

Модуль для генерации случайных чисел:

```
import random
```

Цикл для генерации случайных чисел:

```
for i in range(100000):
```

Получаем случайную букву из строки `q`:

```
    k=random.choice(q)
```

Добавим в FF случайную букву из строки `q`:

```
    FF.append(k)
```

Создаём строку pyt из списка FF:

```
pyt=''.join(FF)
```

Запишем строку в файл:

```
F.write(pyt)
```

Обязательно закроем файл, иначе он не будет создан!

```
F.close()
```

Откройте в Блокноте файл python.txt. Ну и как вам его размерчик? Удалось найти хотя бы одну запись "питон"?

Упражнение 19.2.2

Ну а здесь всё понятно из комментариев в программе:

```
F=open('python.txt',encoding="utf-8" )# снова открываем файл
TXT=F.read() #Запишем строку в переменную TXT
if 'питон' in TXT: #есть "питон" в TXT
    python=TXT.count ('питон') #Сосчитаем "питонов" в строке
    print(python) #напечатаем результат подсчёта
else: #а вдруг запись не найдена?
    print ('Записи "питон" не найдено')
F.close ()  #Закроем файл
```

После запуска этой программы на экране читаем, например (число записей "питон" у вас может оказаться другим – в зависимости от работы предыдущей программы):

```
запись "питон" найдена 31 раз
```

Упражнение 19.2.3

Модифицируем программу из предыдущего упражнения. Запишем имя открываемого файла в виде переменной (да-да, так можно), и при этом допустим ошибку (имя файла записано без кавычек)

```
try:
    A=python.txt
    F=open(A,encoding="utf-8")# снова открываем файл
TXT=F.read() #Запишем строку в переменную TXT
    if 'питон' in TXT: #есть "питон" в TXT
        python=TXT.count ('питон')  #Сосчитаем "питонов" в строке
        print('запись "питон" найдена',python,'раз') #напечатаем
результат подсчёта
    else: #запись не найдена
        print ('Записи "питон" не найдено')
    F.close ()  #Закроем файл
except FileNotFoundError: # файл не найден
    print ('файл с именем ',A,'не найден')
except NameError:# некорректное имя
    print ('Похоже, имя файла записано без кавычек')
```

В этом случае на экране:

Похоже, имя файла записано без кавычек

А теперь запишем 2-ю строчку программы с другой ошибкой:

```
A='pyton.txt'
```

(неверное имя – надо *python*, а у нас *pyton*)

На экране:

```
файл с именем pyton.txt не найден
```

Вот ещё почему имя файла удобно записывать в виде

переменной – мы видим имя файла на экране!

Ну а если всё сделано правильно:

```
A='python.txt'
```

то на экране:

```
запись "питон" найдена 36 раз
```

ну или:

```
запись "питон" найдена 30 раз
```

Ну а такое:

```
Записи "питон" не найдено
```

мы вряд ли увидим – всё-таки в среднем на 100 000 букв слово "питон" должно составиться 32 раза (это называется "математическое ожидание").

Упражнение 19.3

Программа получилась очень большая! Поэтому она разбита на несколько функций.

Эта функция печатает таблицу:

```
def table(R):
    print ('\nФАМИЛИЯ          ','Имя              ','№ телефона
')
    lenR=len(R)
    for j in range(0,lenR,NumS): # построчно выводим данные
#   Методом .format выравниваем данные под заголовками
        S1='{0:<16}'.format(R[j]) #фамилия
        S2='{:<15}'.format(R[j+1]) #имя
        S3='{:^16}'.format(R[j+2]) #телефон
```

```
        print(S1,S2,S3)
    return
```

Эта функция ожидает ответа – будем ли вводить новую запись:

```
def YesNo():
    while True:
        InP=input('\n           введём новую запись? (да/нет)')
        if InP.isalpha():
            InP=InP[0].lower()#сделаем первую букву ввода
строчной
                        # и далее проверяем только первую букву
            if InP=='l' or InP=='д': #введено да
                return 'yes'
            elif InP=='y' or InP=='н': #введено нет
                return 'no'
            else:
                print ('непонятно. Вводите только да/нет')
```

Эта функция выводит текст `Txt`, ожидает ввод и очищает полученную запись от всего, что не является цифрой:

```
def Dig_In (Txt):
    while True:
        IN=input(Txt)
        IN=Chisto(IN) # очищаем от всего, кроме цифр
        if IN.isdigit():
            return IN
        else:
            print('Непонятно. Введите № заново!')
```

Эта функция выводит текст `Txt`, ожидает ввод и проверяет, что введены только буквы:

```
def Let_In(Txt):
    while True:
        IN=input(Txt)
```

```
        if IN.isalpha():
            return IN
        else:
            print('Вводите только буквы!')
```

Функция очистит полученный номер Num от нецифровых знаков:

```
def Chisto(Num):
    Chist='' #очищенная строка
    for k in Num:
        if k.isdigit(): #в очищенную строку помещаем только цифры
            Chist=Chist+k
    return Chist
```

Собственно, программа:

```
Tel=open('telephone.txt',encoding="utf-8")# открываем файл для
чтения
TELE=Tel.read() #считаем файл построчно
T123=TELE.split()# разобьём строку на подстроки по разделителю-
пробелу
Tel.close() # закроем файл - он пока не нужен
NumS=3 # Кол-во столбцов в таблице
```

Эта часть программы и занимается общением с человеком:

```
while True:
    LenA=len(T123)
    Abo=0 # Индикатор. Показывает, что абонент был найден
    print ('введите фамилию, имя или № телефона абонента')
    print('или введите "*" для печати списка абонентов')
    print ('для выхода введите + или =')
    Intel=input('ожидаю ввод: ')
    if Intel.isalpha(): # Введена буквица, значит, надо найти
абонента #по фамилии (1-ый столбец) или имени (2-ой столбец)
        InTel=Intel.title() #Первые буквы сделаем заглавными
        InTel=InTel.replace('ё','е') #ё заменяем на е
```

```
        for i in range(LenA): # Проверяем содержимое строки
            if i%3!=2:
                if InTel==T123[i]:
                    if i%3==0: # выясним, что совпало - фамилия
или имя
                        for j in range(3): #Фамилия совпала
                            print(T123[i+j], end=' ')
                        print()
                    else:
                        for j in range(3): #имя совпало
                            print(T123[i+j-1], end=' ')
                        print()
                    Abo=1 #Абонент найден

    else: #это не имя и не фамилия
        if Intel=='=' or Intel=='+': # ВЫХОД ?
            break
        else:
            if Intel=='*': #печать таблицы
                table(T123)
#это 3-я колонка с номерами телефонов
        Chist=Chisto(Intel)# очищаем от всего, кроме цифр
        for i in range(2,LenA,NumS):
            if Chist==T123[i]: # ищем совпадения по 3-ей колонке

                for j in range(3): # № совпал
                    print(T123[i+j-2], end=' ')
                print()
                Abo=1 #Абонент найден
    if Abo==1:#Абонент найден => вернёмся в начало
        continue
#сюда попадаем, если абонент не найден
    if YesNo()=='no':
        continue # не желаем вводить нового
# вводим нового абонента
    else:
        Fam=Let_In('Фамилия абонента ')
        T123.append(Fam)
```

```
        Name=Let_In('Имя абонента ')
        T123.append(Name)
        Numer=Dig_In('№ телефона абонента ')
        T123.append(Numer)
# сохраняем новую таблицу
    Tele123=' '.join(T123)
    Rvr=open('telephone.txt','w',encoding="utf-8")# открываем
файл для записи

    Rvr.write(Tele123) #записываем
    Rvr.close()#закрываем
print ("+++++    До новых встреч!    +++++")
```

Запускаем, нажимаем * и получаем таблицу (в процессе отладки я кое-что успел набрать):

ФАМИЛИЯ	Имя	№ телефона
Петров	Петр	88893654786
Ершов	Ёрш	788113654786
Bond	James	78880000007
Император	Цезарь	88851234567

Дальше можно самому пополнять список.

Программа очень большая, и это ещё без дополнительного столбца, например, с кличками, с возможностью поиска абонента по сокращённому имени (Михаил=Михась=Мишук=Миша) или сортировкой по столбцам... Фактически – это демонстрационная программа, показывающая, как выглядят и что умеют современные программы.

Упражнение 20.1

Очень простые задачи:

```
A=range(0,11,2)
Qwadro=list(map(lambda x: x**2,A))
```

```
print(A,Qwadro)
```

На экране:

[0, 4, 16, 36, 64, 100]

Вторая задача чуть посложнее, тут главное в конце 4 скобки ")" не забыть поставить:

```
print(list(map(lambda y: y**3, range (-3,10,3))))
```

вывод:

[-27, 0, 27, 216, 729]

Упражнение 20.2

Здесь тоже всё просто, можно даже через лямбда функцию всё проделать:

```
A = [1, 2, 3, 4, 5]
B = (10, 9, 8, 7, 6)
C = [4.4, -2.2, 31.51, 20.6, 10]
D = [5.1, 2.2, 18.3, -2.12, 22]
Q = list(map(lambda a, b, c, d: a*b/(c + d), A, B, C, D))
print (Q)
```

Запускаем, любу... Стоп, что случилось? Что это? На экране много красных строчек, из которых самые понятные – две последние:

Q=list(map(lambda a,b,c,d: a*b/(c+d), A, B, C, D))
ZeroDivisionError: float division by zero

Ясно-понятно: опять произошло деление на 0... У нас C[1]=-2.2, и D[1]=2.2, при сложении получается 0... Решение простое: если

C+D равно 0, то... То присваиваем Q значение (текстовое, конечно): "деление на 0!". Конечно, придётся написать небольшую пользовательскую функцию:

```
def Assemb(a, b, c, d):
    if c+d==0:
        return 'деление на 0!'
    return round(a*b/(c+d),4)

A=[1, 2, 3, 4, 5]
B=(10, 9, 8, 7, 6)
C=[4.4, -2.2, 31.51, 20.6, 10]
D=[5.1, 2.2, 18.3, -2.12, 22]
Q=list(map(Assemb, A, B, C, D))
print (Q)
```

Вот теперь порядок, на экране видим:

```
[1.0526, 'деление на 0!', 0.4818, 1.5152, 0.9375]
```

Упражнение 20.3

Аккуратно переписываем оба варианта программы, добавляем модуль time(). Так как Питон работает очень быстро, зададим большой массив градусов – из 900 элементов.

```
from math import *
from time import time
grad=[]
for i in range (900):
    grad.append(i) # организуем массив градусов
T=time() # засекаем начало работы
Rad=list(map(radians,grad)) #градусы -> радианы
Sin=list(map(sin,Rad)) #радианы -> синусы
RndSin=list(map(lambda Rrr: round(Rrr, 3), Sin))
```

```
ZIPPO=dict(list(zip(grad,RndSin))) #сцепим градусы с синусами
T1=time()-T
print('Время работы с преобразованием на каждом этапе',T1)

T=time()# вновь засекаем начало работы
Rad=map(radians,grad)#градусы -> радианы
Sin=map(sin,Rad) #радианы -> синусы
RndSin=list(map(lambda Rrr: round(Rrr, 3), Sin))
ZIPPO=list(zip(grad,RndSin))#сцепим градусы с синусами
T2=time()-T
print('Время работы с преобразованием только на последнем
этапе',T2)
```

Запускаем – наблюдаем результат:

Время работы с преобразованием на каждом этапе 0.002038240432739258
Время работы с преобразованием только на последнем этапе 0.0

Какой Питон шустрый! Ладно, увеличим размеры списка `grad()` до 10 000 элементов:

Время работы с преобразованием на каждом этапе 0.009956121444702148
Время работы с преобразованием только на последнем этапе 0.004992246627807617

М-да, ожидаемого ускорения в 100 раз не получилось... Почему? "Темна вода во облацех... [32]" , не всегда можно предугадать скорость выполнения программ...

Упражнение 20.4.1

```
from random import randint
Ff=open('100.txt', 'w') # Создадим новый (или переделаем старый)
файл
Rand=[] #сюда поместим наши случайные числа
```

32 Очередная цитата из Библии. Так говорят о чём-то непонятном

```
NumDig=10_000 #количество чисел
razbros=10_000_000 #разброс случайных чисел
```

Сначала, как обычно, создаём список, куда поместим полученные числа, и задаём константы.

```
for i in range(NumDig):
    x=randint (1,razbros) # получаем случайное число
    Rand.append(x/100) #вот вам числа с 2-мя знаками после запятой
Digi=" ".join(map(str, Rand)) #Преобразуем список чисел в строку
```

А здесь самое сложное. Сначала мы функцией `map()` преобразуем каждое число в строчку, а затем методом `.join()` получаем, наконец, длинную строку для записи в файл.

```
Ff.write(Digi) #записали файл
Ff.close() #закрыли файл
```

Упражнение 20.4.2

Здесь всё понятно из комментариев:

```
F100=open('100.txt')
Digi=F100.read()# Содержимое файла --> в текстовую переменную
DigTxt=Digi.split(' ') # разбиваем на список, состоящий из чисел в виде текста
digs=list(map(float,DigTxt)) # преобразуем текст в числа
cnt=0
Dig_Del=131313 # число, на которое должна делится сумма

for i in range (len(digs)):
    for j in range(i+1,len(digs)):
        X=digs[i]+digs[j] # складываем очередную пару
        if X%Dig_Del==0: #делится на заданное число?
            print (digs[i],digs[j]) # печатаем пару
            cnt+=1 # увеличиваем счётчик
```

```
print('Делится на',Dig_Del,':',cnt,'пары')
F100.close() # закрываем файл
```

На экране, для примера:

70648.55 60664.45
55109.69 76203.31
81606.99 49706.01
Делится на 131313 : 3 пары

Как вы уже давно поняли, при использовании функций из модуля `random` у вас каждый раз будут получаться несколько иные результаты.

Упражнение 21.1

У меня получилась вот такая программа:

```
import time
import random
AZ='1234567890qwertyuiopasdfghjklzxcvbnm' #список всех букв
и цифр
print("+++ сейчас мы измерим вашу реакцию +++")
k=random.randint (0,len(AZ)) #Выбираем случайным образом букву
из AZ
time.sleep(3) #ждём 3 сек.
start = time.time()#начало ожидания
txt = input("Нажмите клавишу "+str( AZ[k])+'  ')
if txt==str(AZ[k]):
    end = time.time() #нажата верная клавиша
    reac = end - start #время ожидания
    print("Время вашей реакции: ", reac," секунд.") #вывод
else: # неверная клавиша
    print ('мимо')
```

Ну что ж, проверьте свою же реакцию своей программой.

Упражнение 21.2

Вот такая несложная программа получилась

```
import time
import random
Q=random.randint(8,15) # генерация числа из интервала
t1=time.time() # засекаем начало
a=input('через '+str(Q)+' секунд нажмите ENTER')
t2=time.time() # засекаем момент нажатия
T=round(t2-t1-Q,1) #определяем отклонение от заданного времени
if T < 0:
    print ("Поторопились на", -T, 'секунд')
else :
    print ("задержались на", T, 'секунд')
```

Сделайте несколько замеров своего ощущения времени.
А теперь определите свой темперамент: если вы сильно торопитесь – холерик, немного торопитесь – сангвиник, немного задерживаетесь – меланхолик, сильно задерживаетесь – флегматик.

Упражнение 21.3

Займёмся сначала функцией согласования числительного с существительным "день". В качестве аргументов функция получает целое число дней и … слово "день", которое надо согласовать с числом. Этой же функции в дальнейшем предстоит согласовать слова "месяц" и "год". Дадим ей звучное имя YeMeDay (сокращение от "год-месяц-день" для знающих английский)

```
def YeMeDay(n, k):
```

какое слово будем согласовывать:

```
    if k=='день':
```

Мне не хочется повторять код из упражнения 5.5 – любая сложная программа имеет несколько вариантов написания. Поэтому сделаем немного по-другому: сначала сразу проверим попадание числа дней в самый сложный интервал от 20 до 10:

```
    if 20 > n > 10:
        return 'дней'
```

Дальше пойдёт проще; если числительное оканчивается на цифру 1, то пишем "день":

```
    elif n%10==1:
        return  'день'
```

А вот если оканчивается на цифры 2,3,4 – пишем "дня":

```
    elif (4 >= (n%10) >= 2):
        return 'дня'
    return 'дней'
```

Аналогично напишем фрагмент для слова "год" (не забудьте, что в русском языке не пишут "20 годов", а пишут "20 лет" – наверное, чтобы замучить иностранцев, изучающих русский язык). А для слова "месяц" фрагмент будет ещё и проще – 13 месяцев бывает только в сказках.

В результате функция выглядит так:

```
def YeMeDay(n,k):
    if k=='день':
        if 20>n>10:
            return 'дней'
        elif n%10==1:
            return 'день'
        elif (4>=(n%10)>=2):
```

```
            return 'дня'
        return 'дней'
    elif k=='месяц':
        if n==1:
            return 'месяц'
        if 4>=n>=2:
            return 'месяца'
        return 'месяцев'
    else:
        if 20>n>10:
            return 'лет'
        elif n%10==1:
            return 'год'
        elif (4>=(n%10)>=2):
            return 'года'
        return 'лет'
```

Оставим функцию в покое и перейдём к написанию собственно программы, которая получилась довольно солидная:

```
Day=int(input("Введите день вашего рождения: "))
Mon=int(input("Введите месяц вашего рождения (числом): "))
Year=int(input("Введите год вашего рождения: "))
```

Здесь всё просто: получаем от пользователя день, месяц и год рождения.

```
DM=[31,28,31,30,31,30,31,31,30,31,30,31]
```

Список DM содержит число дней в каждом месяце.

```
MonDig={'Jan':1,'Feb':2, 'Mar':3,'Apr':4,'May':5,'Jun':6,'Jul':7,
'Aug':8,'Sep':9,'Oct':10,'Nov':11,'Dec':12}
```

А вот словарь MonDig ("месяц-число") превращает сокращённое английское название месяца в его порядковый номер в году.

```
sec = time.time()#время в секундах с начала эпохи
Ltime = time.ctime(sec) # местное время в виде строки
MonStr=Ltime[4:7] # сокращённое название месяца на английском
MON=MonDig[MonStr] #с помощью словаря MonDig переведём месяц
в число
DAY=int(Ltime[8:10]) #вычленяем из строки день
YEAR=int(Ltime[20:25]) #... и год (числом)
```

Начинаем расчёты. Из номера года на данный момент вычтем год рождения:

```
Ye=YEAR-Year #сколько лет прошло
```

Из номера месяца вычитаем месяц вашего рождения:

```
Mo=MON-Mon #сколько месяцев
```

А здесь нас может ожидать сюрприз! Если вы родились в ноябре, а сейчас на дворе февраль, то число прошедших месяцев станет отрицательным (2-11=-9... А между февралём и ноябрём 3 месяца! Значит, добавляем к результату 12 месяцев (то есть 1 год), и, соответственно, уменьшаем число лет на 1.

```
if Mo<0: #число месяцев получилось отрицательным
    Mo=12+Mo #добавим 12
    Ye=Ye-1 #и уменьшим число лет на 1
```

С числом дней может получиться то же самое (родились 6 числа, а сегодня 22, поэтому 6-22=-16). Значит добавим число дней, содержащихся в прошлом месяце, и уменьшим число месяцев на 1.

```
Da=DAY-Day #сколько дней прошло
if Da<0: #число дней отрицательное
    Da=DM[Mo-1]+Da #добавим число дней в прошедшем месяце
    Mo=Mo-1 #уменьшим число месяцев на 1
```

Число месяцев опять может стать отрицательным; а вот нулевое число месяцев допустимо. Если вы родились 2 марта 2023 года, а сегодня 16 марта того же года, то ведь вам сейчас 0 месяцев, 12 дней. Да-да и 0 лет!:)

```
    if Mo<0: # число месяцев получилось отрицательным
        Mo=12+Mo #добавим 12
        Ye=Ye-1 #и уменьшим число лет на 1
print ('Вам сейчас',Ye,YeMeDay(Ye,'год'),Mo,YeMeDay(Mo,'месяц'),
Da,YeMeDay(Da,'день'))
```

Проверяем:

- **Сегодня, допустим, 23 января 2023 года. Вы родились 1 января 2023. Ответ:**

Вам сейчас 0 лет 0 месяцев 22 дня

- **Вы родились 31 декабря 2022 года. Ответ:**

Вам сейчас 0 лет 0 месяцев 23 дня

- **Вы родились 23 декабря 2022 года. Ответ:**

Вам сейчас 0 лет 1 месяц 0 дней

- **Вы родились 24 января 2022 года. Ответ:**

Вам сейчас 1 год 0 месяцев 1 день

Работает нормально! Проверяйте дальше сами!

Проверили? Заметили, что программу использовать неудобно – уточнили свой возраст, и программа завершила работу.

А ведь ещё надо узнать возраст папы, мамы, жены, сёстры... (также ещё есть свояки, племянники, ну и, конечно, Месси

с Овечкиным... [33]). Следовательно, надо преобразовать программу в соответствии с требованиями функционального программирования – оформить как функцию, а вызывать из бесконечного цикла!

Готово, переделал, это было несложно:

```
import time
def YeMeDay(n,k):#согласуем числительное с существительным
    if k=='день':
        if 20>n>10:
            return 'дней'
        elif n%10==1:
            return k
        elif (4>=(n%10)>=2):
            return 'дня'
        return 'дней'
    elif k=='месяц':
        if n==1:
            return k
        if 4>=n>=2:
            return 'месяца'
        return 'месяцев'
    else:  # согласуем года
        if 20>n>10:
            return 'лет'
        elif n%10==1:
            return k
        elif (4>=(n%10)>=2):
            return 'года'
        return 'лет'


def Counter():
    Day=input("Введите день вашего рождения: ")
    if Day.isdigit(): #введено число
        Day=int(Day)
```

33 Свояк – муж сестры жены, племянник – сын брата или сестры, Лионель Месси – выдающийся аргентинский футболист современности, Александр Овечкин – выдающийся российский хоккеист современности

```
    else: # введена буква
        print ('       Сеанс окончен. До свидания!')
        return 0
    Mon=int(input("Введите месяц вашего рождения (числом): "))
    Year=int(input("Введите год вашего рождения: "))
    DM=[31,28,31,30,31,30,31,31,30,31,30,31]
    MonDig={'Jan':1,'Feb':2, 'Mar':3,'Apr':4,'May':5,'Jun':6,
'Jul':7, 'Aug':8,'Sep':9,'Oct':10,'Nov':11,'Dec':12}

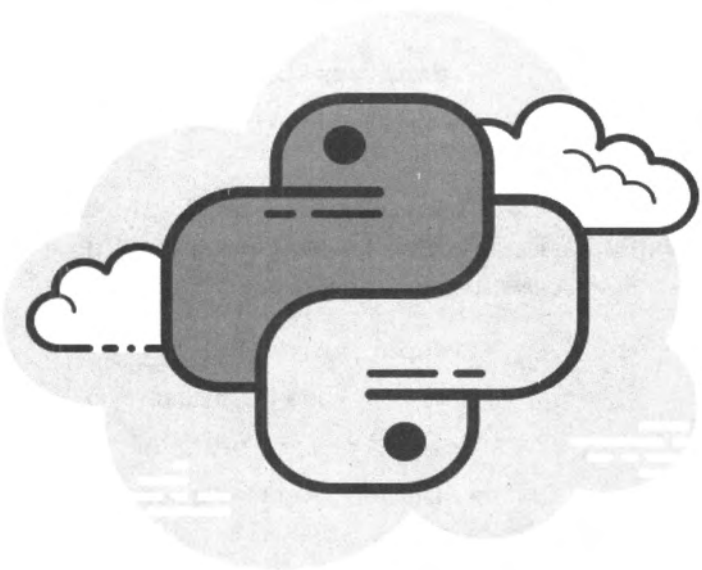
    sec = time.time()#время в секундах с начала эпохи
    Ltime = time.ctime(sec) # местное время в виде строки
    #print("Местное время:", Ltime)
    MonStr=Ltime[4:7] # сокращённое название месяца на английском
    MON=MonDig[MonStr] #с помощью словаря MonDig переведём месяц
в число
    DAY=int(Ltime[8:10]) #вычленяем из строки день
    YEAR=int(Ltime[20:25]) #... и год (числом)
    print (MON,DAY,YEAR)
    Ye=YEAR-Year #сколько лет прошло
    Mo=MON-Mon #сколько месяцев
    if Mo<0: #число месяцев получилось отрицательным
        Mo=12+Mo #добавим 12
        Ye=Ye-1 #и уменьшим число лет на 1
    Da=DAY-Day #сколько дней прошло
    if Da<0: #число дней отрицательное
        Da=DM[Mo-1]+Da #добавим число дней в прошедшем месяце
        Mo=Mo-1 #уменьшим число месяцев на 1
        if Mo<0: # число месяцев получилось отрицательным
            Mo=12+Mo #добавим 12
            Ye=Ye-1 #и уменьшим число лет на 1
    print ('Вам сейчас',Ye,YeMeDay(Ye,'год'),Mo,YeMeDay(Mo,'месяц'),
Da,YeMeDay(Da,'день'))
    return 1


#  Собственно программа начинается лишь отсюда
print ('''  Программа вычисляет количество лет, месяцев и дней
со дня вашего рождения.
        Для окончания работы введите любую букву \n''')
```

```
while True: #бесконечный цикл
    if Counter()==0: #окончание работы - введена буква
        break
```

Приложение 1.

ОСНОВНЫЕ МЕТОДЫ ОБРАБОТКИ СТРОК



По традиции, назначим некие подопытные строки:

```
>>>deca='я мыслю, - значит, существую'[1]
>>>  BEATLES='ALL YOU NEED IS LOVE'[2]
>>>b2c='b2c'[3]
```

capitalize() – делает прописной первую букву строки.

```
>>>deca.capitalize
'Я мыслю, - значит, существую'
```

center(width,[fill]) – выдаёт отцентрированную строку, по краям которой стоит символ `fill` (по умолчанию это пробел):

```
>>>deca.center(36)
'  я мыслю, - значит, существую  '
>>>deca.center(36,':')
'::::я мыслю, - значит, существую::::'
```

1 Афоризм Р. Декарта - великого французского математика, философа, физика и физиолога

2 В переводе с английского "Всё, что тебе нужно – это любовь" – знаменитая песня группы "The Beatles"

3 b2c (по-английски пишется business-to-consumer) — это модель бизнеса, в которой компания продаёт товар частному лицу. К примеру, это - продуктовые магазины, развлекательные центры, кафе и рестораны, кинотеатры

count(str, [start],[end]) – выдаёт количество вхождений подстроки **str** в диапазоне [start,end] (а по умолчанию от начала до конца).

```
>>>deca.count('ю')
2
>>>deca.count('ю',20,30)
1
```

find (str, [start], [end]) – выдаёт номер первого вхождения подстроки **str** в диапазоне [start,end] (а по умолчанию от начала до конца), выдаёт -1, если подстрока не найдена.

```
>>>deca.find('ю',20,30)
27
```

index (str, [start],[end]) – выдаёт номер первого вхождения подстроки **str** в диапазоне [start,end] (а по умолчанию от начала до конца), выдаёт *ValueError*, если подстрока не найдена. Это полезно, если нам надо обойти этот случай с помощью оператора **try**.

```
>>>BEATLES.index('O')
5
>>>BEATLES.index('O',5)
5
>>>BEATLES.index('O',6)
17
```

isalnum() – проверяет, состоит ли строка из цифр или букв. Выдаёт *True* в таком случае:

```
>>>deca.isalnum()
```

False

(в строке есть запятая и тире)

```
>>>BEATLES.isalnum()
```

False

(оказывается, пробелы за буквы не считаются! Вот так сюрприз от имени Питона!)

```
>>>b2c.isalnum()
```

True

isalpha() – проверяет, состоит ли строка из букв. Выдаёт *True* в таком случае:

```
>>>deca.isalpha()
```

False

(в строке есть запятая и тире)

isdigit() – проверяет, состоит ли строка из цифр. Выдаёт *True* в таком случае:

```
>>>deca.isdigit()
```

False

(в строке вообще нет ни одной цифры)

```
>>>'021'.isdigit()
```

True

islower() – состоит ли строка из символов в нижнем регистре.

```
>>>deca.islower()
```

True

```
>>>b2c.islower()
```

True

То есть цифры игнорируются

isupper() – состоит ли строка из символов в верхнем регистре.

```
>>>deca.isupper()
False
>>>BEATLES.isupper()
True
```

istitle() – начинаются ли слова в строке с заглавной буквы.

```
>>>BEATLES.istitle()
False
```

ЧТО? Да здесь ведь вся строка состоит из заглавных букв! Я что-то не так понимаю? Совершенно верно – не так. Здесь слова должны только начинаться с заглавных букв, а дальше должны идти обычные строчные буквы:

```
>>>'All You Need Is Love'.istitle()
True
>>>'All you need is Love'.istitle()
False
```

join(<последовательность>) – самый необычный метод, в нём всё наоборот. Он отвечает за объединение *списка строк* с помощью определенного разделителя. Метод собирает из последовательности любого типа строку, разделяя элементы стоящим впереди разделителем:

<строка>=<разделитель>.join(<последовательность>)

```
>>>'.'.join(deca)
'я. .м.ы.с.л.ю.,. .-. .з.н.а.ч.и.т.,. .с.у.щ.е.с.т.в.у.ю'
```

Ну, собрать из строки строку большого ума не надо.

```
>>>m='1','2','3'
```

```
>>>''.join(m)
'123'
```

Разделитель может и вовсе отсутствовать.

```
>>>'+'.join(b2c+BEATLES)
'b+2+c+A+L+L+ +Y+O+U+ +N+E+E+D+ +I+S+ +L+O+V+E'
```

ljust(width, char) – делает длину строки не меньшей width, заполняя символы справа знаком **char**.

```
>>>BEATLES.ljust(32,'*')
'ALL YOU NEED IS LOVE**********'
```

lower() – переводит все символы строки в нижний регистр.

```
>>>BEATLES.lower()
'all you need is love'
```

lstrip([str]) – удаляет указанные (по умолчанию пробелы) символы в начале строки.

```
>>>BEATLES.lstrip('ALL')
' YOU NEED IS LOVE'
```

replace (old, new, [max]) – заменяет подстроки *old* на подстроки *new*. Необязательный параметр устанавливает максимальное число замен.

```
>>>deca.replace('с','ч',2)
'я мычлю, - значит, чуществую'
```

rfind (str, [start], [end]) – выдаёт номер первого с конца вхождения подстроки **str** в диапазоне [start,end] (а по умолчанию от начала до конца), выдаёт -1, если подстрока не найдена. То есть поиск начинается справа, о чём говорит буква "r" в названии метода ("right" – правый).

```
>>>deca.rfind('ю')
27
```

rjust(width, char) – делает длину строки не меньшей width, заполняя символы слева знаком char.

```
>>>BEATLES.rjust(32,'+')
'++++++++++++ALL YOU NEED IS LOVE'
```

rstrip([str]) – удаляет указанные (по умолчанию пробелы) символы в конце строки.

```
>>>deca.rstrip('ю')
'я мыслю, - значит, существу'
```

Но:

```
>>>deca.rstrip('y')
'я мыслю, - значит, существую'
```

То есть, если в самом конце строки символ не нашёлся – ничего не делаем

split([<разделитель> [,<лимит>]]) – разбивает строку на подстроки по указанному разделителю. Если скобки пустые – разделяем по пробелу. Второй параметр <лимит> ограничивает количество подстрок.

```
>>>BEATLES.split()
['ALL', 'YOU', 'NEED', 'IS', 'LOVE']
>>>BEATLES.split('L')
```

['A', '', ' YOU NEED IS ', 'OVE']

```
>>>BEATLES.split('O',1)
```

['ALL Y', 'U NEED IS LOVE']

Как видите, разделитель исчезает из строки.

strip([str]) – удаляет указанные (по умолчанию пробелы) символы в начале и конце строки.

```
>>>'абракадабра'.strip('а')
```

'бракадабр'

swapcase() – заглавные буквы заменяются на строчные и наоборот.

```
>>>Deca='Я Мыслю, - Значит, Существую'
>>>Deca.swapcase()
```

'я мЫСЛЮ, - зНАЧИТ, сУЩЕСТВУЮ'

title() – делает заглавной первую букву каждого слова.

```
>>>deca.title()
```

'Я Мыслю, - Значит, Существую'

upper() – делает заглавными все символы строки.

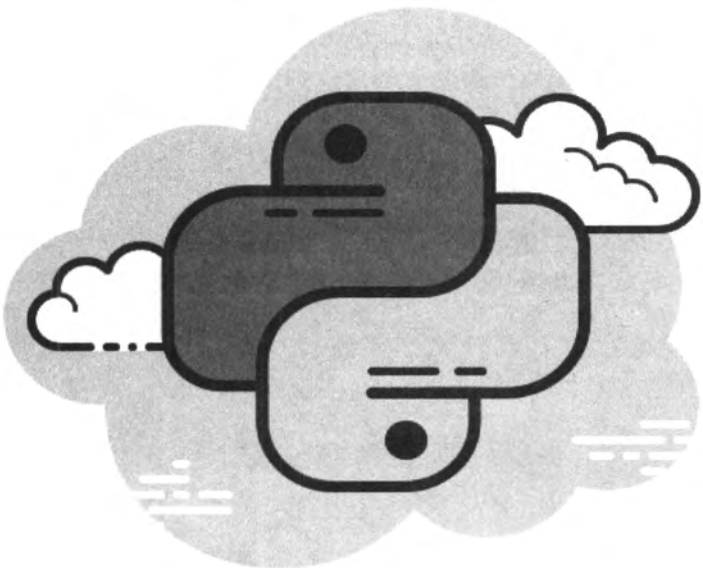
```
>>>deca.upper()
```

'Я МЫСЛЮ, - ЗНАЧИТ, СУЩЕСТВУЮ'

Самое главное, что методы строк, в отличие от методов обработки списков, *не изменяют саму строку*, **а потому результат выполнения надо записывать к какую-нибудь (можно в ту же самую) переменную.**

Приложение 2.

ОСНОВНЫЕ МЕТОДЫ ОБРАБОТКИ СПИСКОВ



Введём подопытные списки:

```
Dig=[11,12,13]
Wrd=['слово','дело']
mini=[-1.1,-12.3]
```

append(q) – добавляет элемент ***q*** в конец списка.

```
>>> Dig.append('ещё')
>>> Dig
[11, 12, 13, 'ещё']
```

clear() – очищает список.

```
>>>Many=[1,2,3,4,5,6,7,8,9]
>>>Many
[]
```

count (Q) – сообщает нам количество элементов ***Q*** в последовательности:

```
>>>C123=[1,2,1,3,13,1,1]
>>>C123.count(1)
4
```

extend(Q) – добавляет список ***Q*** в конец списка. Главное, здесь можно добавить сразу список, а не 1 элемент, как в append(q).

```
>>>mini.extend(Wrd)
>>>mini
[-1.1, -12.3, 'слово', 'дело']
```

То же самое можно сделать, используя **+** (конкатенацию)

index(x, [start [, end]]) – возвращает положение первого элемента со значением ***x*** (при этом поиск ведётся от *start* до *end*)

```
>>>Wrd.index('дело')
1
>>>Many=[1,2,3,4,5,6,7,8,9]
>>>Many.index(5)
4
```

insert(i, Q) – вставляет элемент Q на i-ое место.

```
>>>Wrd.insert(1,'и')
>>>Wrd
['слово', 'и', 'дело']


[-1.1, -12.3, 'дело']
```

pop([i]) – удаляет i-ый элемент в списке. Если индекс не указан – удаляет последний.

```
>>>Wrd.pop(1)
'и'
>>>Wrd
['слово', 'дело']
```

remove(Q) – удаляет *первый* элемент в списке, имеющий значение ***Q***. Происходит ошибка ValueError, если такого элемента не существует.

```
>>>mini.remove('слово')
>>>mini
[-1.1, -12.3, 'дело']
```

reverse() – переворачивает список.

```
>>>Dig.reverse()
>>>Dig
['ещё', 13, 12, 11]
```

sort([reverse=True]) – сортирует список по возрастанию, для сортировки в обратном порядке укажите параметр **reverse=True.**

```
>>>C123.sort()
>>>C123
[1, 1, 1, 1, 2, 3, 13]
>>>C123=[1,2,1,3,13,1,1]
>>>C123.sort(reverse=True)
>>>C123
[13, 3, 2, 1, 1, 1, 1]
```

Вот только сортировать этот метод умеет лишь списки, состоящие из однородных элементов:

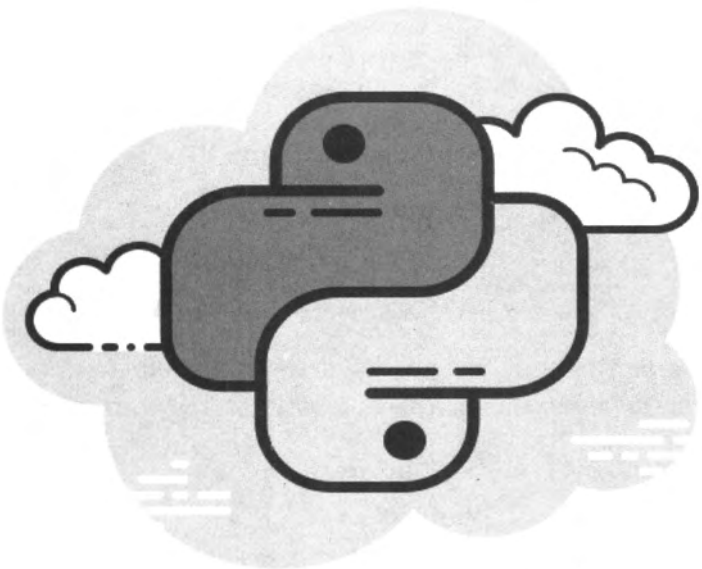
```
>>>Wrd.sort()
>>>Wrd
['дело', 'слово']
>>>Dig.sort()
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    Dig.sort()
TypeError: '<' not supported between instances of 'str' and 'int'
```

Перевожу: "не умею я сравнивать строки и числа".

Самое главное, что методы списков, в отличие от строковых методов, *изменяют сам список*, **а потому результат выполнения не нужно записывать в эту переменную.**

Приложение 3.

ОСНОВНЫЕ МЕТОДЫ ОБРАБОТКИ СЛОВАРЕЙ



Где подопытная строка? Вот она:

```
Lat={1:"Finis", 2:"coronat", 3:"opus"}
```
final

clear() – очищает словарь.

copy() – возвращает копию словаря.

```
>>>LT1=Lat.copy()
>>>LT1
{1: 'Finis', 2: 'coronat', 3: 'opus'}
```

get(key[, default]) – возвращает значение ключа; если указанного значения ключа нет, выводит значение **default** (по умолчанию **None**).

```
>>>Lat.get(2)
'coronat'
```

final — Это последняя сноска в книге, поэтому очередная латинская пословица переводится как "Конец – всему делу венец"

Если вы уверены, что нужный ключ есть в словаре, удобнее, конечно, обращаться к значению прямо по ключу:

```
>>>Lat[2]
'coronat'
```

keys() – выдаёт ключи в словаре, причём в виде объекта:

```
>>>Lat.keys()
dict_keys([1, 2, 3])
```

Как "прочитать" содержимое объекта, мы уже знаем – функцией `list`:

```
>>>Q=Lat.keys()
>>>list(Q)
[1, 2, 3]
```

pop(key) – удаляет ключ и возвращает его значение. Если ключа нет – получим ошибку **KeyError** и остановленную программу.

```
>>>Lat.pop(2)
'coronat'
>>>Lat
{1: 'Finis', 3: 'opus'}
```

А теперь попытаемся удалить уже несуществующий ключ:

```
>>>Lat.pop(2)
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    Lat.pop(2)
KeyError: 2
```

Получаем ошибку.

update([other]) – обновляет/дополняет словарь, добавляя пары (ключ, значение) из **other**. Существующие ключи перезаписываются.

Можно вот таким, затейливо-запутанным способом:

```
>>>Lat.update([(4,'Gloria')])
>>>Lat
{1: 'Finis', 2: 'coronat', 3: 'opus', 4: 'Gloria'}
```

А можно проще, просто добавляя куски словаря:

```
>>>Lat.update({5:'victory bus'})
>>>Lat
{1: 'Finis', 2: 'coronat', 3: 'opus', 4: 'Gloria', 5: 'victory bus'}
```

Перезапишем ошибочный ключ:

```
>>>Lat.update({5:'victoribus'})
>>>Lat
{1: 'Finis', 2: 'coronat', 3: 'opus', 4: 'Gloria', 5: 'victoribus'}
```

values() – выдаёт значения в словаре, причём в виде объекта:

```
>>>Q=Lat.values()
```

А как "прочитать" содержимое объекта, мы уже знаем – функцией `list`:

```
>>>list(Q)
['Finis', 'coronat', 'opus', 'Gloria', 'victoribus']
```

Список использованных источников информации:

- Кольцов Д.М., Дубовик Е.В. Справочник PYTHON. Кратко, быстро, под рукой – Спб.: Наука и Техника, 2021.
- Сузи Р.А. Python. – СПб.: БХВ-Петербург, 2002.
- Лутц М. Изучаем Python, 3-е издание – Пер. с англ. – СПб.: Символ_Плюс, 2009.
- Любанович Билл Простой Python. Современный стиль программирования. — СПб.: Питер, 2016.
- Мюллер, Джон Пол Python для чайников, 2-е изд. : Пер. с англ. - СПб. : ООО "Диалектика", 2019.
- python-script.com
- pythonru.com
- habr.com
- pythononline.ru
- https://icons8.ru/icon/plJdjOoL6KfU/%D0%BF%D0%B8%D1%82%D0%BE%D0%BD
- https://www.flaticon.com/free-icon/python_3049800?term=python&related_id=3049800

Книги по компьютерным технологиям, медицине, радиоэлектронике

Уважаемые авторы!

Приглашаем к сотрудничеству по изданию книг по **IT-технологиям, электронике, медицине, педагогике.**

Издательство существует в книжном пространстве более 20 лет и имеет большой практический опыт.

Наши преимущества:

- Большие тиражи (в сравнении с аналогичными изданиями других издательств);
- Наши книги регулярно переиздаются, а автор автоматически получает гонорар с *каждого* издания;
- Индивидуальный подход в работе с каждым автором;
- Лучшее соотношение цена-качество, влияющее на объемы и сроки продаж, и, как следствие, на регулярные переиздания;
- Ваши книги будут представлены в крупнейших книжных магазинах РФ и ближнего зарубежья, библиотеках вузов, ссузов, а также на площадках ведущих маркетплейсов.

Ждем Ваши предложения:

тел. **(812) 412-70-26** / эл. почта: **nitmail@nit.com.ru**

Будем рады сотрудничеству!

Для заказа книг:

- **интернет-магазин: www.nit.com.ru / БЕЗ ПРЕДОПЛАТЫ по ОПТОВЫМ ценам**
  - более 3000 пунктов выдачи на территории РФ, доставка 3-5 дней
  - более 300 пунктов выдачи в Санкт-Петербурге и Москве, доставка 1-2 дня
  - тел. (812) 412-70-26
  - эл. почта: nitmail@nit.com.ru

- **магазин издательства:** г. Санкт-Петербург, пр. Обуховской обороны, д.107
  - метро Елизаровская, 200 м за ДК им. Крупской
  - ежедневно с 10.00 до 18.30
  - справки и заказ: тел. (812) 412-70-26

- **крупнейшие книжные сети и магазины страны**

  Сеть магазинов «Новый книжный» тел. (495) 937-85-81, (499) 177-22-11

- **маркетплейсы ОЗОН, Wildberries, Яндекс.Маркет, Myshop и др.**

Иванов С. С.

Начнем PYTHON

Просто о сложном

Группа подготовки издания:

Зав. редакцией компьютерной литературы: *М. В. Финков*

Редактор: *Е. В. Финков*

Корректор: *А. В. Громова*

12+

ООО "Издательство Наука и Техника"

ОГРН 1217800116247, ИНН 7811763020, КПП 781101001

192029, г. Санкт-Петербург, пр. Обуховской обороны, д. 107, лит. Б, пом. 1-Н

Подписано в печать 05.06.2023. Формат 70x100 1/16.

Бумага офсетная. Печать офсетная. Объем 23 п.л.

Тираж 2000. Заказ 6914.

Отпечатано с готового оригинал-макета

ООО «Принт-М», 142300, М.О., г.Чехов, ул. Полиграфистов, д.1

ИВАНОВ С. С.

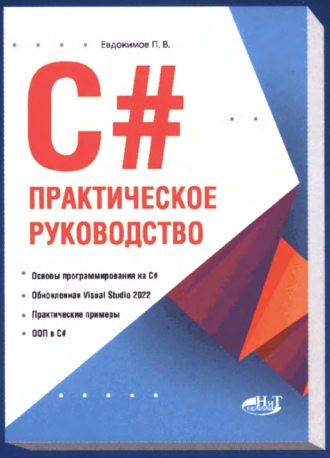
Начнем.Python

Просто о сложном

5 причин купить эту книгу:

- Легкий и понятный язык автора
- Рассмотрены все ключевые аспекты Python
- Практические примеры и задания к каждой главе
- Самостоятельное создание программ на Python
- Не требует для прочтения навыков программирования

издательство НАУКА и ТЕХНИКА рекомендует



ISBN 978-5-907592-23-0

«Издательство Наука и Техника» г. Санкт-Петербург
Для заказа книг: т. (812) 412-70-26
E-mail: nitmail@nit.com.ru
Сайт: nit.com.ru

nit.com.ru