

Д. Златопольский

# **ОСНОВЫ** **программирования** **на** **языке** **Python**

**Второе издание**

Златопольский Д. М.

# **Основы программирования на языке Python**

2-е издание



Москва, 2018

**УДК 373.167.1:004.42+004.42(075.3)**

**ББК 32.973.721**

**367**

**367** Златопольский Д. М.

Основы программирования на языке Python. 2-е изд. – М.: ДМК  
Пресс, 2018. – 396 с.: ил.

**ISBN 978-5-97060-641-4**

Книга представляет собой учебник по программированию на языке Python. Она написана простым языком, приводятся типичные ошибки начинающих программировать, и даётся ряд полезных советов. Рассмотрены основные типовые задачи и методы их решения с подробными комментариями. Во второе издание добавлено Приложение 3 с ответами к заданиям и задачами по разработке программ.

Издание будет полезно школьникам и студентам, а также учителям средних школ и преподавателям вузов и колледжей и всем, кто начинает изучать программирование с помощью языка Python или уже имеющих небольшой опыт программирования на другом языке.

УДК 373.167.1:004.42+004.42(075.3)

ББК 32.973.721

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-5-97060-641-4

© Златопольский Д. М., 2018

© Оформление, издание ДМК Пресс, 2018



# Оглавление

<b>Введение .....</b>	<b>7</b>
<b>Глава 1. Понятия «алгоритм» и «программа» .....</b>	<b>8</b>
<b>Глава 2. Python. Первые шаги .....</b>	<b>13</b>
<b>Глава 3. Вывод информации на экран.....</b>	<b>21</b>
<b>Глава 4. Переменные величины. Ввод данных в программу. Инструкция присваивания .....</b>	<b>31</b>
<b>Глава 5. Варианты действий в программе .....</b>	<b>45</b>
5.1. Два варианта действий .....	45
5.2. Один, но не обязательный вариант действий .....	54
5.3. Три и более вариантов действий .....	59
<b>Глава 6. Повторение действий в программе .....</b>	<b>65</b>
6.1. Инструкция for .....	65
6.2. Инструкция while.....	75
6.3. Преобразование одной инструкции цикла в другую .....	83
<b>Глава 7. Программируем простейшие игры .....</b>	<b>92</b>
7.1. Игра «Чет или нечет?» .....	92
7.2. Игра «Кубик» .....	94
7.3. Игра «Отгадай число» .....	95
7.4. Игра «Карты» .....	96
7.5. Проверка знания таблицы умножения.....	98
7.6. Игра «Предметы на столе» .....	98
<b>Глава 8. Повторение повторений .....</b>	<b>102</b>
<b>Глава 9. «Обрабатываем» натуральное число.....</b>	<b>113</b>
9.1. Выделение цифр .....	113
9.2. Определение $m$ -й справа цифры числа.....	114
9.3. Определение $m$ -й слева цифры числа .....	115
9.4. Определение суммы цифр числа .....	116
9.5. Определение максимальной цифры числа.....	116
9.6. Определение минимальной цифры числа .....	117

9.7. Определение номера максимальной цифры числа при счете справа налево .....	117
9.8. Определение номера минимальной цифры числа при счете справа налево .....	118
<b>Глава 10. Типовые задачи обработки набора чисел .....</b>	<b>126</b>
10.1. Суммирование всех чисел набора.....	126
10.2. Суммирование чисел набора, которые обладают некоторыми свойствами (удовлетворяют некоторому условию) .....	127
10.3. Подсчет количества чисел набора, которые обладают некоторыми свойствами .....	128
10.4. Определение среднего арифметического тех чисел набора, которые обладают некоторыми свойствами .....	129
10.5. Определение порядкового номера некоторого значения в заданном наборе .....	131
10.6. Определение максимального значения в наборе чисел .....	132
10.7. Определение порядкового номера максимального значения в наборе чисел .....	134
10.8. Определение максимального значения тех чисел набора, которые удовлетворяют некоторому условию .....	135
10.9. Нахождение второго по величине максимального числа набора .....	136
10.9.1. Поиск числа, которое стояло бы на предпоследнем месте, если бы числа набора были отсортированы по убыванию .....	137
10.9.2. Нахождение числа набора, больше которого только максимальное.....	139
10.10. Нахождение количества максимальных элементов набора .....	140
10.11. Нахождение третьего максимума.....	141
<b>Глава 11. Работа со строками .....</b>	<b>145</b>
11.1. Общие вопросы .....	145
11.2. Типовые задачи обработки строк .....	148
11.3. Преобразования «число $\leftrightarrow$ строка» .....	163
<b>Глава 12. Использование списков.....</b>	<b>165</b>
12.1. Общие вопросы .....	165
12.2. Заполнение списка значениями.....	166
12.3. Вывод списка на экран.....	171
<b>Глава 13. Типовые задачи обработки списков .....</b>	<b>176</b>
13.1. Расчеты .....	176
13.1.1. Суммирование элементов списка .....	176

13.1.2. Нахождение суммы элементов списка с заданными свойствами (удовлетворяющих некоторому условию) .....	177
13.1.3. Нахождение количества элементов списка с заданными свойствами .....	178
13.1.4. Нахождение среднего арифметического значения элементов списка с заданными свойствами.....	179
13.2. Поиск и отбор нужных элементов .....	181
13.2.1. Вывод на экран элементов с заданными свойствами .....	181
13.2.2. Запись всех элементов списка с заданными свойствами в другой список .....	181
13.2.3. Вывод на экран индексов элементов списка с заданными свойствами.....	182
13.2.4. Поиск индекса первого элемента списка с заданными свойствами.....	182
13.3. Работа с максимальными/минимальными элементами списка.....	183
13.3.1. Определение индекса максимального элемента списка .....	184
13.3.2. Определение количества максимальных/минимальных элементов списка .....	185
13.3.3. Нахождение второго по величине (второго максимального или второго минимального) значения списка .....	186
13.4. Перестановки элементов .....	188
13.4.1. Обмен местами двух элементов списка .....	188
13.4.2. Удаление элемента из списка .....	188
13.4.3. Циклический сдвиг элементов списка влево .....	191
13.4.4. Вставка элемента в список .....	192
13.4.5. Циклический сдвиг элементов списка вправо .....	194
13.4.6. Перестановка всех элементов списка в обратном порядке .....	195
13.5. Проверка соответствия списка в целом некоторому условию .....	197
13.5.1. Проверка факта наличия в списке элемента с заданными свойствами (удовлетворяющего некоторому условию).....	197
13.5.2. Проверка факта наличия в списке элемента с заданным значением .....	203
13.5.3. Проверка того факта, что все элементы списка соответствуют некоторому условию.....	203
13.5.4. Проверка списка на упорядоченность .....	203
13.6. Задача «Слияние (объединение) списков» .....	204
<b>Глава 14. Использование словарей.....</b>	<b>208</b>
14.1. Общие вопросы .....	208
14.2. Создание словаря.....	209
14.3. Обращение к отдельному элементу словаря .....	210
14.4. Перебор элементов словаря.....	211
14.5. Некоторые другие средства для работы со словарями.....	211
14.6. Частотный словарь .....	212



14.7. Словари со значениями разных типов .....	213
<b>Глава 15. Использование файлов .....</b>	<b>216</b>
15.1. Общие вопросы .....	216
15.2. Запись информации в файл .....	218
15.3. Чтение информации из файла.....	221
15.4. Изменение файлов .....	232
15.4.1. Запись в файл новой строки.....	232
15.4.2. Замена строки файла.....	233
<b>Глава 16. Об использовании функций.....</b>	<b>235</b>
<b>Приложение 1. Служебные (ключевые) слова языка Python.....</b>	<b>250</b>
<b>Приложение 2. Разрабатываем графический интерфейс программы .....</b>	<b>251</b>
П2.1. Общие вопросы.....	251
П2.2. Создание виджетов .....	255
П2.3. Размещаем виджеты .....	259
П2.4. Доступ к значениям в виджетах .....	262
П2.5. Изменение конфигурации виджетов .....	266
П2.6. Заставляем виджеты работать .....	268
П2.7. Итоги.....	279
П2.8. Задания для самостоятельной работы .....	280
<b>Приложение 3. Ответы к заданиям. Программы решения задач, предложенных для самостоятельной работы .....</b>	<b>284</b>
<b>Литература.....</b>	<b>395</b>



# Введение

*Когда человек хочет передвинуть гору, он начинает с того, что убирает маленькие камни.*

Восточная мудрость

Эта книга для тех, кто хочет научиться программировать на языке программирования Python. Ее отличие в том, что в ней главное – не описание языка программирования, которое представлено в большинстве книг по программированию. В данной книге рассматриваются особенности разработки программ, описываются методы решения типовых задач, встречающихся при разработке, распространенные алгоритмы, даются советы и т. п. Это учебник, в котором в доступной форме излагаются вопросы, с которыми сталкивается человек, начинающий осваивать этот непростой, но захватывающий и очень интересный процесс – программирование.


В первой части книги (главы 1–6) рассматриваются особенности разработки компьютерных программ и соответствующие инструкции языка Python. После их изучения читатель сможет разработать ряд простейших игр, описанных в главе 7. В завершающей части книги (главы 8–16) рассматриваются основные структуры данных языка Python (строки, списки, словари), вопросы, связанные с работой с файлами, а также с совершенствованием программ на основе использования функций.

По всем темам, рассмотренным в книге, приводятся большое число примеров и задач, подробная методика их решения и соответствующие программы с комментариями. Дается ряд полезных советов.

В приложении 2 описаны особенности разработки программ на языке Python с графическим пользовательским интерфейсом.

В приложении 3 приводятся ответы к заданиям и программы решения задач, предложенных в книге для самостоятельной работы.





# Глава 1.

## Понятия «алгоритм» и «программа»

*Алгоритм решения задачи* – точное описание порядка действий, которые надо выполнить для решения задачи.

Приведем ряд примеров.

Вот старинная задача: «Однажды крестьянину понадобилось перевезти через реку волка, козу и капусту. У крестьянина есть лодка, в которой может поместиться, кроме самого крестьянина, только одно существо или предмет – или волк, или коза, или капуста. Если крестьянин оставит без присмотра волка с козой, то волк съест козу; если крестьянин оставит без присмотра козу с капустой – коза съест капусту. Как крестьянину перевезти на другой берег все свое имущество в целости и сохранности?»

Чтобы решить задачу, крестьянин должен действовать так:

- 1) погрузить на лодку козу;
- 2) перевезти ее на другой берег;
- 3) выгрузить ее;
- 4) вернуться;
- 5) погрузить на лодку волка;
- 6) перевезти его на другой берег;
- 7) выгрузить его;
- 8) погрузить на лодку козу;
- 9) вернуться с ней на первый берег;
- 10) выгрузить ее;
- 11) погрузить на лодку капусту;
- 12) перевезти ее на другой берег;
- 13) оставить капусту на этом берегу;
- 14) вернуться на первый берег;
- 15) погрузить на лодку козу;
- 16) перевезти ее на другой берег;

- 17) выгрузить козу;
- 18) остаться на втором берегу.

Возможен и второй вариант решения задачи (найдите его). Самостоятельно составьте также алгоритм решения такой задачи: «К реке подошла группа из 20 солдат, которым нужно переправиться на другой берег. Река глубокая и бурная, и ее без лодки переплыть невозможно. Солдаты увидели двух мальчиков с лодкой. Лодка такова, что в ней размещается только один солдат, только один мальчик или только два мальчика. Как солдатам переправиться?»

Можно говорить об алгоритме приготовления торта в домашних условиях (только в этом случае мы алгоритм называем «рецептом»).

Пример из математики. Представьте, что товарищ принес вам чертеж прямоугольника и просит сообщить ему площадь этой фигуры. Какие действия вы должны выполнить, чтобы решить поставленную перед вами задачу? Вот ответ:

- 1) узнать (измерить) длину одной из сторон прямоугольника и запомнить (или записать) ее;
- 2) узнать (измерить) длину второй стороны фигуры и запомнить (или записать) ее;
- 3) рассчитать площадь, перемножив два найденных значения;
- 4) сообщить результат товарищу.

Каждый алгоритм рассчитан на какого-то *исполнителя* – человека (группу людей) или устройство (робот и др.), который выполняет действия, описанные в алгоритме.

## Задание для самостоятельной работы

Представьте, что на стержень *A* надеты три диска:



Рис. 1.1

Задача состоит в том, чтобы перенести диски со стержня *A* на стержень *C*, используя стержень *B* как промежуточный. При этом должны соблюдаться три условия:

- 1) за один «ход» можно переносить лишь один диск;
- 2) нельзя класть больший диск на меньший;
- 3) снятый диск нельзя отложить в сторону – он должен быть надет на один из стержней.

Опишите алгоритм решения задачи.

Понятие *«программа решения задачи»* возникает, когда речь идет о решении задачи на компьютере. Программа решения задачи – это алгоритм решения данной задачи, записанный в виде (на языке), «понятном» данному компьютеру. А какой язык «понимает» компьютер? В нем, как в техническом устройстве, информация может быть представлена в двоичном виде – в виде последовательности условных нулей и единиц. Значит, и описание действий, которые нужно выполнить для решения (в программировании их называют «командами»), должно поступать в компьютер в двоичном виде. Поэтому в первых электронных компьютерах программы представляли из себя последовательность двоичных чисел<sup>1</sup>:

1010 1101 0101 1000 ...

Представляете, как трудно было программисту найти нужное место в программе, чтобы что-то изменить или добавить, найти и исправить ошибку? Чтобы устранить этот недостаток, были разработаны так называемые «языки программирования высокого уровня» (ЯПВУ)<sup>2</sup> – Фортран, Алгол, Бейсик и др. Программы на ЯПВУ оформляются в привычном человеку<sup>3</sup> виде: числа записываются как десятичные, а команды и другие служебные слова – на естественном (пусть и иностранном, но достаточно понятном) языке: PRINT, IF, BEGIN и т. п. Однако при этом получается противоречие – человеку-программисту удобно разрабатывать и читать программу, но компьютер такую программу не «поймет»!

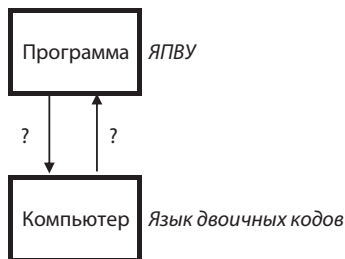


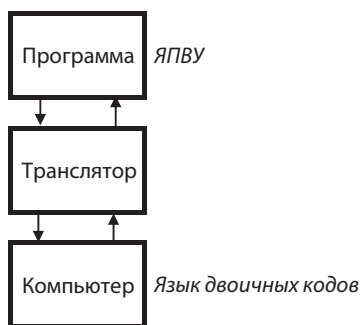
Рис. 1.2

<sup>1</sup> Так как числа в двоичной системе «длинные», то для записи программ использовалась также восьмеричная или шестнадцатеричная система счисления.

<sup>2</sup> Мы не рассматриваем использование для разработки программ языков ассемблера, имевшее место до применения ЯПВУ.

<sup>3</sup> ЯПВУ поэтому так и называются, что они, так сказать, «ориентированы» на человека, в отличие от машиноориентированных языков (языков двоичных кодов и ассемблеров).

Как же поступить, чтобы компьютер мог выполнять программу на ЯПВУ? А так, как можно общаться двум людям, один из которых знает только китайский язык, а второй – только русский: нужен переводчик! Поэтому во всех современных системах программирования предусмотрен транслятор – системная программа, осуществляющая перевод прикладной программы, написанной программистом на ЯПВУ, в язык машинных кодов и ее выполнение (см. рис. 1.3)<sup>4</sup>.



**Рис. 1.3**

Существуют два вида трансляторов:

- 1) интерпретаторы;
- 2) компиляторы.

Интерпретатор читает очередную команду программы и сразу ее выполняет, не переводя всю программу в машинный код.

Компилятор читает всю программу целиком, делает ее перевод и создает законченный вариант программы на «машинном языке», который затем и выполняется.

Вопрос о преимуществах и недостатках каждого варианта транслятора здесь обсуждать не будем<sup>5</sup>, а скажем, что язык программирования Python<sup>6</sup>, как правило, работает в режиме интерпретации.

В качестве примера программы на Python приведем программу решения задачи расчета площади любого прямоугольника:

```
#Ввод размеров прямоугольника
a = int(input('Введите длину прямоугольника в см'))
b = int(input('Введите высоту прямоугольника в см'))
#Расчет площади
pl = a * b
```

<sup>4</sup> Транслятор обычно выполняет также диагностику ошибок и др.

<sup>5</sup> Подумайте над этим вопросом.

<sup>6</sup> Язык Python иногда называют языком программирования не просто высокого, а «очень высокого уровня».

```
#Вывод ответа на экран  
print('Площадь прямоугольника равна', pl, 'кв. см')
```

Итак, Python – это язык программирования, на котором разрабатываются программы. Чтобы сделать этот процесс удобным для вас и других программистов, разработаны *системы программирования*, включающие транслятор, текстовый редактор (позволяющий копировать, удалять и перемещать фрагменты программ), справочную систему, средства отладки (поиска ошибок в программе) и др. Имеется несколько вариантов языка Python и систем программирования для него.

В данной книге используется вариант CPython с версией 3.6.0 языка, который распространяется по свободной лицензии Python Software Foundation License.

## Контрольные вопросы

1. Что такое «алгоритм решения задачи»?
2. Какие вы знаете способы записи алгоритма?
3. В чем особенности алгоритма, который называют «программой»?
4. Почему языки программирования высокого уровня так называются?
5. Что такое «транслятор»? Какие функции он выполняет?
6. Какие виды трансляторов вы знаете? В чем особенность каждого вида?
7. Что включает в себя система программирования?

# Глава 2.

## Python. Первые шаги

Итак, вы установили систему программирования с языком Python. Запустите ее: **Пуск** → **Все программы** → **Python 3.6** → **idle (Python 3.6 32-bit)**:

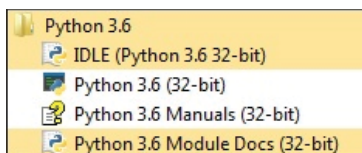


Рис. 2.1

Откроется главное окно **Python Shell** так называемой *интегрированной среды разработки* (Integrated DeveLopment Environment – IDLE)<sup>1</sup>:

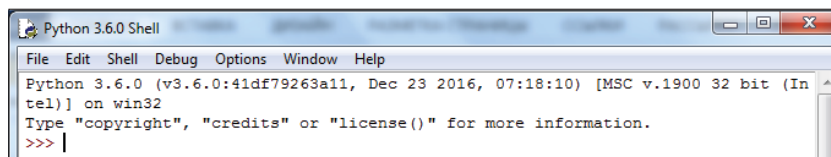


Рис. 2.2. Окно Python Shell

Окно **Python Shell** обеспечивает доступ к интерактивному режиму работы, когда каждая введенная команда сразу выполняется.

Поскольку никаких команд языка Python мы пока не знаем, то будем использовать систему программирования как калькулятор (возможности системы это позволяют).

1.  $2 + 5$
2.  $3 * (5 - 8)$
3.  $2.4 + 3.0/2$

<sup>1</sup> В дальнейшем окно **Python Shell** удобнее вызывать с помощью ярлыка, который надо разместить, например, на рабочем столе. Для размещения ярлыка нужно перетащить строку **idle (Python 3.6 32-bit)** (см. рис. 2.1) на рабочий стол.

Наберите подобные примеры в интерактивном режиме (после `>>>`; в конце каждого примера нажимайте клавишу `<Enter>`). Обратите внимание, что в числах с дробной частью в качестве разделителя используется точка, а не запятая. Набранные команды сразу выполняются, и результат выводится на экран.

В интерактивном режиме можно также писать и выполнять очень простые программы. Но для написания сложных программ используется другой режим работы – программный, когда записывается вся программа и при запуске выполняется целиком (предварительно она сохраняется в файле на диске (что удобно для повторного выполнения)). Программу на Python часто называют «скриптом».

Мы будем говорить главным образом о программном режиме.

Чтобы перейти в программный режим, нужно в меню **File** выбрать пункт **New File** или одновременно нажать клавиши `<Ctrl+N>`. Появится окно для разработки программы (окно редактора):

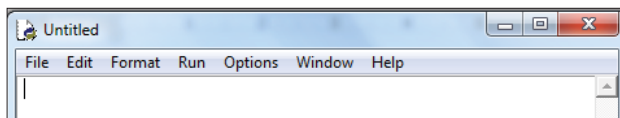


Рис. 2.3

Теперь мы можем написать свою первую программу на Python. В большинстве книг по программированию первая программа решает задачу вывода на экран приветствия «Здравствуй, мир!». Так же поступим и мы.

Напомним, что программа, как и алгоритм решения задачи, состоит из команд, которые надо выполнить для решения задачи. Эти команды в языке Python называются «инструкциями».

Вот текст программы:

```
print('Здравствуй, мир!')
```

Вы, конечно, поняли, что для вывода на экран некоторого текста в языке Python используется инструкция `print()`. Ее особенности подробно описаны в следующей главе.

После начала набора текста программы в окне редактора в заголовке окна слово `Untitled` будет «окружено» символами `«*»`:

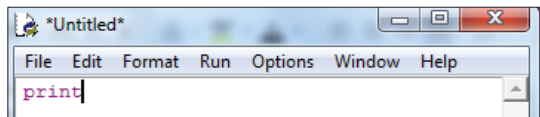


Рис. 2.4



Это говорит о том, что программа в окне еще не записана в файл на диске.

Для сохранения программы нужно в меню **File** выбрать пункт **Save As...** или одновременно нажать клавиши **<Ctrl+S>**. Появится окно для выбора имени файла с программой и папки, в которой он будет размещен:

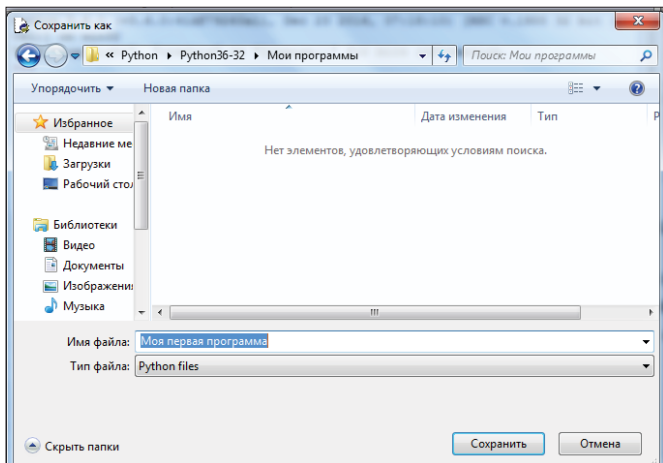


Рис. 2.5

Назовем файл **Моя первая программа**. Система добавит к этому имени расширение **.py**. По умолчанию файлы с программами размещаются в папке с файлами системы программирования. Удобнее создать в ней отдельную папку для собственных программ. Найти папку с файлами системы программирования Python можно, щелкнув правой кнопкой мыши на ярлыке (см. сноску 1) и выбрав в появившемся меню пункт **Расположение файла**.

Чтобы выполнить программу, нужно нажать функциональную клавишу **<F5>**. Результат выполнения программы появится в окне **Python Shell**:

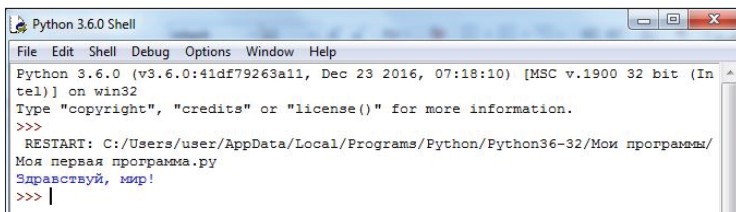


Рис. 2.6

Если перед выполнением программы не сохранить ее, система предложит это сделать:

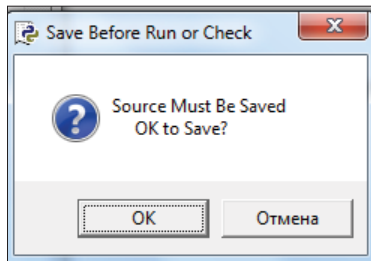


Рис. 2.7

Для выхода из интерактивного режима работы, то есть из окна **Python Shell**, следует закрыть это окно – система вернется в окно редактора.

Добавим в программу еще одну инструкцию так, как показано на рис. 2.8:

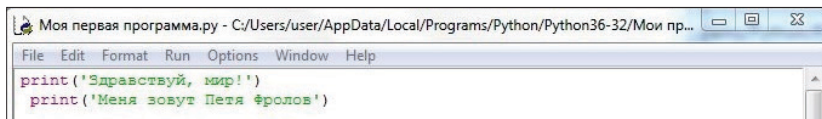


Рис. 2.8

Сохраним новый вариант программы (используя меню **File** → **Save** или клавиши **<Ctrl+S>**) и попробуем выполнить ее (**<F5>**) – появится сообщение об ошибке:

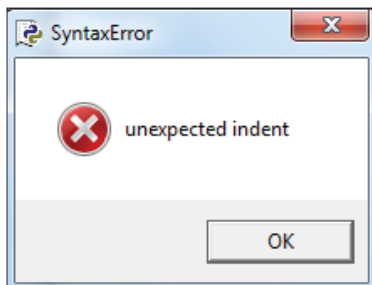


Рис. 2.9

Причина в том, что вторая инструкция записана с некоторым смещением относительно первой. В программах на языке Python все инструкции должны быть записаны с одним и тем же отступом. Исключения составляют так называемые «составные инструкции», которые

содержат другие инструкции и каким-либо образом управляют их выполнением. Обычно составные инструкции записываются в несколько строк:

```
инструкция1:
    инструкция2
    инструкция3
...
```

При этом все внутренние инструкции записываются с одним и тем же отступом относительно «наружной» инструкции. Отступ можно сделать, нажав клавишу **<Tab>**. В конце первой строки составной инструкции указывается символ «:».

В некоторых простых случаях составная инструкция может быть записана в программе в одну строку:

```
инструкция1: инструкция22
```

«Внутренние» инструкции также могут быть составными:

```
инструкция1:
    инструкция2
    инструкция3
...
инструкция8
инструкция9:
    инструкция10
    инструкция11:
        инструкция12
        инструкция13
...
```

Кроме инструкций, в программах принято писать так называемые «комментарии» — тексты, помогающие читающему программу (в том числе автору программы) понять ее особенности.

В приведенной в главе 1 программе строки, начинающиеся символом «#», — это и есть комментарии. Комментарии могут быть записаны и после инструкции:

```
p1 = a * b    #Расчет площади
```

При выполнении программы транслятор комментарии игнорирует.

---

<sup>2</sup> В одну строку могут быть записаны и инструкции «одного уровня» (при этом их надо разделять точкой с запятой):

```
инструкция1; инструкция2
```

Так рекомендуется поступать только в случаях, когда инструкции логически связаны и их немного.

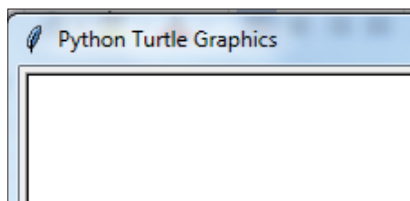
В данной книге вы увидите в программах много комментариев (хотя опытные программисты говорят, что «много комментариев не бывает»...).

## Дополнение

В Python имеется возможность не только проводить расчеты и работать с текстами, но и создавать графические изображения<sup>3</sup>. Самые простые графические возможности обеспечивает использование исполнителя «turtle» («черепаха»). Этот исполнитель представляет собой «перо», оставляющее след на плоскости рисования. «Перо» можно поднять, тогда при перемещении след оставаться не будет. Кроме того, для «пера» можно установить толщину и цвет.

Приведенный ниже фрагмент программы создает графическое окно (рис. 2.10), в котором происходит рисование, и помещает «черепаху» (в виде ►) в исходное положение.

```
import turtle #Инструкция импорта модуля с командами исполнителя
turtle.reset() #Инициализация исполнителя
```



**Рис. 2.10. Верхняя левая часть окна Python Turtle Graphics**

Управление исполнителем «черепаха» осуществляется следующими основными командами (инструкциями)<sup>4</sup>:

Команда	Назначение	Пример
<code>forward(n)</code>	Передвижение вперед (в направлении острия стрелки ►) на <i>n</i> точек	<code>turtle.forward(100)</code>
<code>backward(n)</code>	Передвижение назад на <i>n</i> точек	<code>turtle.backward(100)</code>
<code>up()</code>	Поднятие «пера», чтобы не оставалось следа его при перемещении	<code>turtle.up()</code>

<sup>3</sup> Конечно, перечислены лишь основные возможности языка.

<sup>4</sup> Имеются и другие команды.

Команда	Назначение	Пример
<code>down()</code>	Опускание «пера», чтобы при перемещении оставался след (рисовались линии). По умолчанию «перо» опущено	<code>turtle.down()</code>
<code>right(k)</code>	Поворот направо (по часовой стрелке) на <i>k</i> единиц (по умолчанию на <i>k</i> градусов). В начале программы «черепаха» «смотрит» на правую часть окна (на восток)	<code>turtle.right(75)</code>
<code>left(k)</code>	Поворот налево (против часовой стрелки) на <i>k</i> единиц	<code>turtle.left(45)</code>
<code>goto(x, y)</code>	Перемещение «пера» в точку с координатами <i>x</i> , <i>y</i> в системе координат окна рисования. При этом ориентация «черепахи» не меняется	<code>turtle.goto(50, 20)</code>
<code>width(n)</code>	Установка толщины «пера» (в <i>n</i> точек экрана)	<code>turtle.width(3)</code>
<code>tracer(flag)</code>	Включение ( <i>flag</i> = 1) и выключение ( <i>flag</i> = 0) режима отображения «черепахи». По умолчанию включен. При выключенном режиме отображения рисование происходит значительно быстрее, чем при включенном	<code>turtle.tracer(0)</code>
<code>clear()</code>	Очистка области рисования	<code>turtle.clear()</code>

Например, программа для рисования квадрата размером 40 имеет вид:

```
import turtle
turtle.reset()
turtle.forward(40)  #Можно копировать, перемещать и удалять
turtle.right(90)    #любые фрагменты программы
turtle.forward(40)
turtle.right(90)
turtle.forward(40)
turtle.right(90)
turtle.forward(40)
turtle.right(90)
```

## Задание

Напишите программу, в которой на экране получается изображение:

- а) прямоугольника высотой 50 и шириной 100 точек экрана;
- б) правильного<sup>5</sup> шестиугольника;
- в) равностороннего треугольника.

---

<sup>5</sup> Правильный многоугольник – это выпуклый многоугольник, у которого все стороны между собой равны и все углы между смежными сторонами равны.



# Глава 3.

## Вывод информации на экран

Для вывода информации на экран в программах на языке Python используется инструкция `print()`. В скобках указывается то, что нужно вывести. Например, чтобы вывести на экран приветствие, надо записать:

```
print('Привет!')
```

или

```
print("Привет!")
```

то есть текст (последовательность символов) указывается в кавычках.

В результате выполнения инструкции на экран будут выведены все символы, указанные в кавычках, включая начальные и конечные пробелы.

Можно также указывать в скобках:

- число:

```
print(5)
print(-2)
print(3.14)
```

- имя переменной величины<sup>1</sup>:

```
print(a)
print(x1)
print(perimetr)
```

- арифметическое или логическое выражение (о них будет рассказано ниже).

В программах на языке Python могут использоваться также объекты, над которыми выполняются некоторые действия (методы). Например, к последовательности символов может быть применен метод `upper()`, преобразующий все буквы в их написание в верхнем регист-

---

<sup>1</sup> О переменных будет рассказано в главе 4.



ре («прописными буквами»). Чтобы эти действия были выполнены (был применен метод), следует записать так:

```
<имя объекта>.<имя метода>
```

Например:

```
famil.upper()
```

Такую запись (которую называют «точечной») также можно использовать для вывода на экран:

```
print(famil.upper())
```

Результат выполнения программы для каждого из указанных случаев записи инструкции показан в таблице:

Что указано в скобках	Пример	На экран будет выведено	Пример
1. Текст	<code>print('Привет!')</code>	Текст без кавычек, включая возможные начальные и конечные пробелы	Привет!
2. Число	<code>print(-2)</code>	Соответствующее число	-2
3. Имя переменной величины	<code>print(x1)</code>	Значение величины	273
4. Выражение	<code>print(a * b)</code>	Значение выражения	1024
5. Метод	<code>print(famil.upper())</code>	Результат применения метода	ЛУКИН

Можно указывать несколько значений, в том числе разного типа, разделяя их запятой. Например, в программе решения задачи расчета площади и периметра любого прямоугольника может быть использован следующий вариант инструкции:

```
print(1, '. Площадь прямоугольника равна', p1, 'кв. см')
```

в котором указаны число, два текста и имя переменной величины. В результате выполнения программы на экран будет выведено примерно следующее:

```
1. Площадь прямоугольника равна 42 кв. см
```

Видно, что между указанными в инструкции `print()` значениями (будем называть их «список вывода») выводится также один пробел.

Этот разделитель можно изменить на любой другой символ (или последовательность символов). Для этого после списка вывода нужный символ-разделитель указывается как параметр `sep` инструкции, например:

```
print(<список вывода>, sep = ' , ')
```

Можно использовать в качестве разделителя «пустой» символ (`' '`):

Пример	На экран будет выведено
<pre>print(a, '+', b, '=', c, sep = ' ') #a и b - заданные числовые значения #c - их сумма</pre>	2+3=5

Во всех перечисленных случаях каждая новая инструкция `print()` выводит соответствующие значения на следующей строке. Чтобы исключить это, необходимо указать другой параметр этой инструкции – `end`, задав его равным «пустому» символу:

```
print(<список вывода>, end = ' ')
```

Конечно, можно указывать и оба параметра:

```
print(<список вывода>, sep = ' , ', end = ' ')
```

или

```
print(<список вывода>, end = ' ', sep = ' , ')
```

Для вывода пустых строк следует использовать инструкцию `print()` без списка вывода:

Пример	На экран будет выведено
<pre>print('1.') print() print('2.') </pre>	1. 2.

Вывод на экран вещественных чисел (могут быть с дробной частью) имеет особенности. В результате выполнения инструкций

```
print(1/3)
```

и

```
a = 1
b = 3
print(a/b)
```

на экран будет выведено следующее:

```
0.3333333333333333
```

Дело в том, что операция деления (знак операции «/») дает вещественный результат даже в случаях, когда она проводится над целыми числами, в том числе в случаях  $25/5$ ,  $48/12$  и т. п.

Количество цифр в дробной части можно ограничить. Для этого в инструкции `print()` перед выводимым значением следует записать точку, количество цифр `КолДробн` в дробной части и букву `f` в виде:

```
print('% .3f' % <значение>)2
```

или

```
print("% .2f" % <значение>)3
```

где `<значение>` — выводимое значение.

Примеры:

Программа	На экран будет выведено															
<pre>a = 234.193 print('% .1f' % a)</pre>	<p>234.2</p> <table><tr><td></td><td>2</td><td>3</td><td>4</td><td>.</td><td>2</td></tr><tr><td>Позиция экрана слева</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr></table>		2	3	4	.	2	Позиция экрана слева	1	2	3	4	5	6		
	2	3	4	.	2											
Позиция экрана слева	1	2	3	4	5	6										
<pre>a = -234.123 print('% .2f' % a)</pre>	<p>-234.12</p> <table><tr><td>-</td><td>2</td><td>3</td><td>4</td><td>.</td><td>1</td><td>2</td></tr><tr><td>Позиция экрана слева</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table>	-	2	3	4	.	1	2	Позиция экрана слева	1	2	3	4	5	6	7
-	2	3	4	.	1	2										
Позиция экрана слева	1	2	3	4	5	6	7									

Видно, что:

- при «усечении» чисел проводится округление;
- знак «+» не выводится (но позиция для него предусмотрена).

Это так называемый «форматированный вывод» с помощью инструкции `print()`.

Если указанное в инструкции значение `КолДробн` больше фактического количества цифр в дробной части выводимого значения, то справа будут выведены дополнительные нули:

<sup>2</sup> Между первыми двумя символами `' %` пробела быть не должно. В противном случае он также будет выведен.

<sup>3</sup> Между первыми символами `" %` пробела быть не должно.



Программа	На экран будет выведено													
<pre>a = 1/2 print('% .3f' % a)</pre>	<div>0.500</div> <div><table><tr><td></td><td>0</td><td>.</td><td>5</td><td>0</td><td>0</td></tr><tr><td>Позиция экрана слева</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr></table></div>		0	.	5	0	0	Позиция экрана слева	1	2	3	4	5	6
	0	.	5	0	0									
Позиция экрана слева	1	2	3	4	5	6								

Несколько слов о параметре <значение>. Это должно быть имя переменной:

```
a = 1
b = 3
c = a/b
print("% .1f" % c)
```

или конкретное значение:

```
print('% .2f' % 0.567743)
```

или арифметическое выражение в скобках:

```
print('% .1f' % (1/b))
print('% .1f' % (a/3))
print('% .1f' % (a/b))
print('% .1f' % (1/3))
```

Можно установить также общее количество позиций экрана для вывода числа. Это количество **ОбщКол** указывается в формате вывода (между символами '%' или '%') до точки:

```
print('% 7.3f' % <значение>)
```

или

```
print("% 6.2f" % <значение>)
```

Значение **ОбщКол** включает в себя цифры дробной части (**Кол-Дробн**), разделитель-точку и знак числа.

Примеры:

Программа	На экран будет выведено													
<pre>a = 234.193 print('% 6.1f' % a)</pre>	<div>234.2</div> <div><table><tr><td></td><td>2</td><td>3</td><td>4</td><td>.</td><td>2</td></tr><tr><td>Позиция экрана слева</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr></table></div>		2	3	4	.	2	Позиция экрана слева	1	2	3	4	5	6
	2	3	4	.	2									
Позиция экрана слева	1	2	3	4	5	6								

Программа	На экран будет выведено														
<pre>b = -895.451 print('% 7.2f' % b)</pre>	<div><div>-895.45</div><div><table><tr><td>-</td><td>8</td><td>9</td><td>5</td><td>.</td><td>4</td><td>5</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table></div><div>Позиция экрана слева</div></div>	-	8	9	5	.	4	5	1	2	3	4	5	6	7
-	8	9	5	.	4	5									
1	2	3	4	5	6	7									

Если фактическое количество цифр в выводимом значении меньше, чем `ОбщКол`, оно будет дополнено слева пробелами (а справа – возможно, нулями):

Программа	На экран будет выведено														
<pre>b = 30/8 print('% 7.3f' % b)</pre>	<div>3.750</div> <div><table><tr><td></td><td></td><td>3</td><td>.</td><td>7</td><td>5</td><td>0</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table></div> <div>Позиция экрана слева</div>			3	.	7	5	0	1	2	3	4	5	6	7
		3	.	7	5	0									
1	2	3	4	5	6	7									

а если больше, то значение `ОбщКол` игнорируется и в целой части выводится фактическое количество цифр:

Программа	На экран будет выведено																													
<pre>b = 30000/7 print('% 7.3f' % b)</pre>	<div>4285.714</div> <div><table><tr><td></td><td>4</td><td>2</td><td>8</td><td>5</td><td>.</td><td>7</td><td>1</td><td>4</td></tr><tr><td>Позиция</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>экрана слева</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></div>		4	2	8	5	.	7	1	4	Позиция	1	2	3	4	5	6	7	8	9	экрана слева									
	4	2	8	5	.	7	1	4																						
Позиция	1	2	3	4	5	6	7	8	9																					
экрана слева																														

Формат вывода можно применить к нескольким переменным:

```
print('% 6.2f' % a, '% 6.2f' % b)
```

Можно также обеспечить форматированный вывод целых чисел. В этом случае в формате вывода указывается только общее количество позиций экрана `ОбщКол`, а также буква `d` (говорящая о том, что нужно вывести целое число в десятичной системе счисления):



Программа	На экран будет выведено										
<pre>b = 5 print('% 10d '% b)</pre>	<div>5</div> <div><table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>5</td></tr></table><div>Позиция экрана слева</div><div>12345678910</div></div>										5
									5		

Инструкция `print()` с параметром `ОбщКол`, как правило, используется для вывода на экран данных в виде таблиц:

21.55	79.34
-36.40	2.70
128.67	-456.28
3.20	66.38

Возможны и другие виды форматирования выводимых на экран данных (выравнивание на строке и т. п.).

Перед форматированным выводом числа (и/или после него) можно записать также тексты:

```
s = a + b
print('Сумма равна', '% 8d % s')

dlina = 8123.5/7
print('% 7.1f % dlina, 'км')
```

## Контрольные вопросы

1. Что можно указывать в скобках в инструкции `print()`? Что будет выведено на экран в том или ином случае?
2. Можно ли указывать в скобках несколько значений одного типа? Что при этом будет выведено на экран между ними? Как изменить этот разделитель?
3. Можно ли указывать в скобках несколько значений разного типа?
4. Что надо сделать, чтобы после выполнения инструкции `print()` следующие данные выводились на той же строке?
5. В чем особенность вывода на экран вещественных значений?
6. Как можно ограничить количество цифр в дробной части вещественного числа при его выводе на экран?
7. Что устанавливает значение `ОбщКол`, о котором рассказывалось выше? В каком случае оно, как правило, используется?

## Задания

1. Определите (не оформляя программу), что будет выведено на экран в результате выполнения следующих инструкций:

```
print(31, 15, end = '')  
print(77)
```

2. Определите (не оформляя программу), что будет выведено на экран в результате выполнения следующих инструкций:

```
print(51, 36)  
print(77, 45, end = '')
```

3. Определите (не оформляя программу), что будет выведено на экран в результате выполнения следующих инструкций:

```
print(25, 86)  
print()  
print(27, 51, end = '')
```

4. Определите (не оформляя программу), что будет выведено на экран в результате выполнения следующих фрагментов программ:

а)

```
a = 5; b = 3  
print('F(', b, ') = ', a, sep = '')
```

б)

```
a = 5; b = 3  
print('f(a)=', '(b)', sep = '')
```

в)

```
a = 5; b = 3  
print('F(', a, ')=(, b, ')', sep = '')
```

## Задачи для разработки программ

1. Получить на экране следующее:

Три числа: 10 5 24

Текст 'Три числа: 10 5 24' не использовать.

2. Получить на экране следующее:

Три числа: 103, 25, 724

Текст 'Три числа: 103, 25, 724' не использовать.

3. Получить на экране следующее:

1  
2  
3



4. Число  $\pi$  примерно равно 3,1415926. Вывести на экран это число с тремя цифрами в дробной части. Текст '3.142' не использовать.

5. Получить на экране следующее:

```
ooooo124ooooo13
oooooo56ooooo355
ooooo587ooooooo8
```

где символом «o» обозначена пустая позиция на экране.

6. Получить на экране следующее:

```
ooooo1.24oooo13.52
ooo3.567ooo-355.1
oooooo8.2ooooo9.18
```

где символом «o» обозначена пустая позиция на экране.

7. Получить на экране следующее:

```
ooooo7.240ooo-43.520
oooo23.500oooo55.107
oooo88.203oo-769.800
```

где символом «o» обозначена пустая позиция на экране.

В заключение заметим, что в последних версиях Python для форматированного вывода данных используется также метод `format()`. Этот метод форматирует значение-аргумент, указанный в круглых скобках, по шаблону (образцу), который указан в кавычках и фигурных скобках:

```
print('{<шаблон форматирования>}'.format(<значение>))
```

Примеры:

```
print('{: 10.2f}'.format(-4/3))
```

```
a = -4/3
print('{: 10.2f}'.format(a))
```

```
z = 123
print('{: 5d}'.format(z))
```

Особенности оформления шаблона вывода:

- 1) между первыми (левыми) символами '{ пробела быть не должно. В противном случае он также будет выведен;
- 2) после двоеточия пробела может не быть; два и более пробелов записывать нельзя;

- 3) перед «правыми» символами } ' пробела быть не должно (между ними пробелы возможны);
- 4) для вещественных чисел в шаблоне указываются значения КолДробн и ОбщКол (или только КолДробн), для целых – ОбщКол.

Метод `format()` также может быть использован и для нескольких значений сразу (список этих значений указывается в скобках через запятую):

```
a = 1/3
b = 1/9
print('{: 7.3f}{: 7.3f}'.format(a, b))

print('{: 5.1f}{: 5.1f}'.format(1/3, 1/7))

x = 317
y = 123
z = 72
print('{: 5d}{: 5d}{: 5d}'.format(x, y, z))

print('{: 5d}{: 5d}{: 5d}'.format(13, 5, 41))
```

При этом между отдельными шаблонами вывода {...} пробелов быть не должно. В противном случае они также будут выведены.

## Задание

Разработайте программы решения приведенных выше задач 4–7 с использованием метода `format()`.



# Глава 4.

## Переменные величины.

### Ввод данных в программу.

### Инструкция присваивания

Переменные величины (или, короче, переменные) – величины, которые при выполнении программы могут принимать различные значения<sup>1</sup>. Эти значения (в любой момент времени – какое-то одно) хранятся в памяти компьютера и могут быть использованы<sup>2</sup>, поэтому говорят, что переменные используются в программе для хранения информации.

Вспомним «математическую» программу из главы 1. В ней ее пользователь должен ввести размеры сторон прямоугольника, которые могут быть любыми; эти значения будут использованы для расчета площади. Значение площади, которое тоже может быть любым, надо запомнить, а затем вывести на экран. Можем сказать, что в этой программе используются три переменные.

Каждая переменная характеризуется именем и типом.

В именах переменных можно использовать буквы, цифры (но имя не может начинаться с цифры) и знак подчеркивания «\_». Желательно давать переменным «говорящие» имена, чтобы можно было сразу понять, зачем нужна та или иная переменная. Например, имя `discr` помогает понять, что эта переменная хранит значение дискриминанта квадратного уравнения.

Строчные и заглавные буквы различаются, то есть переменные с именами `dlina` и `Dlina` – это две разные переменные.

---

<sup>1</sup> Существуют также так называемые «константы» – так в программировании называют величины, которые во время выполнения программы не меняются. В языке Python имеются также «встроенные» константы, например константы логического типа `True` и `False` (см. далее), константа `pi`, равная числу  $\pi$  (см. далее), и др.

<sup>2</sup> О том, как используются значения переменных, вы, наверное, уже поняли из приведенных в главах 1 и 3 примеров их вывода на экран, – по имени (`print(x1)` и т. п.).

Есть набор слов, которые нельзя использовать в качестве имен переменных, так как эти слова «зарезервированы» в языке Python для определенных целей (эти слова называют «зарезервированными», или «служебными», или «ключевыми»). Перечень таких слов приведен в приложении 1.

Основные типы данных в языке Python приведены в таблице:

Тип	Обозначение типа	К нему относятся	Примеры	Примечание
Целый	<code>int</code>	Целые числа (положительные и отрицательные, а также 0)	4, -45, 0, 687	
Вещественный	<code>float</code>	Вещественные числа <sup>1</sup> (могут быть с дробной частью)	1.45, 0.00453, -3.789,	Как уже указывалось в главе 3, в Python разделителем целой и дробной частей вещественного числа является точка
Логический	<code>bool</code>	Величины, которые могут принимать значения <code>True</code> («Истина») или <code>False</code> («Ложь»)		
Строковый	<code>str</code>	Последовательность (строка) символов, в том числе один символ или пустая строка ('')	'Школа', 'красный', 'h', ''	Подробно о работе с переменными типа <code>str</code> будет рассказано в главе 11

<sup>1</sup> Их называют также «числами с плавающей точкой».

Тип величины определяет:

- какие значения может принимать величина (область допустимых значений переменной), например значением переменной типа `str` может быть только строка символов;

- какие операции можно проводить над переменной (множество допустимых операций с величиной); например, над величинами типа `float` можно проводить операции сложения, вычитания, умножения, деления и возведения в степень, а над величинами типа `int` – еще две операции (см. далее);
- какой объем памяти компьютера требуется для хранения значения данной переменной и в каком формате будут храниться данные. Например, переменные типа `float`, как правило, занимают 8 байтов.

Значение переменной хранится в каком-то месте памяти, которое можно смоделировать в виде прямоугольника (ячейки), рядом с которым указано имя этой переменной:



Как значение «попадает» в соответствующий прямоугольник? Это происходит с помощью двух инструкций:

- 1) инструкции присваивания;
- 2) инструкции ввода данных.

Начнем со второй – инструкции<sup>3</sup> `input()`. Она используется для ввода данных в программу в ходе ее выполнения с помощью клавиатуры (как говорят в программировании – «с клавиатуры»). Например, чтобы ввести значение переменной `a` строкового типа, нужно записать в программе:

```
a = input()
```

При выполнении этой строки на экране появится курсор, и система будет ожидать ввода значения с клавиатуры. Когда пользователь введет его и нажмет клавишу **<Enter>**, система запишет это значение в память в переменную `a`. До нажатия клавиши **<Enter>** можно удалять символы с помощью клавиши **<Backspace>**.

Когда приходится вводить значения нескольких величин, целесообразно указать в скобках сообщение-подсказку, чтобы пользователь программы знал, какое значение вводится в тот или иной момент ее выполнения:

---

<sup>3</sup> Как и применительно к `print()`, мы используем термин «инструкция» (предписание, команда), так как это то, что должно быть выполнено программой. С другой стороны, это функция, так как в результате ее выполнения будет получено некоторое значение (строка символов). О том, что такое функция, см. дополнение в конце данной главы.

```
fam = input('Введите фамилию: ')
im = input('Укажите имя: ')
```

Можно также вывести на экран общее сообщение:

```
print('Введите фамилию, а затем имя: ')
fam = input()
im = input()
```

Можно так оформить инструкции `print()` и `input()`, чтобы ввод значения на экране происходил в той же строке, где выведена подсказка (см. главу 3):

```
print('Введите строку символов', end = '')
st = input()
```

## Инструкция присваивания

Инструкция присваивания позволяет изменить (или задать впервые) значение переменной. В общем случае она оформляется так:

*<имя переменной> = <выражение>*

Например, когда присваивается значение величине числового типа (`int` или `float`), инструкция имеет вид:

*<имя переменной> = <арифметическое выражение>*

где *<арифметическое выражение>* – одно или несколько чисел, имен переменных величин или имен функций<sup>4</sup>, соединенных знаками<sup>5</sup> арифметических операций. Примеры:

```
c = 12
m = n
sum = a + b
stepen3 = a ** 3
x = 2 * a - 3.6 * b/c
sred = (a + b)/2
```

Обратим внимание на знаки возведения в степень («\*\*») и деления («/»), а также на использование в последнем примере круглых скобок для изменения порядка действий (в Python, как и в других языках программирования, выражения записываются в строчку<sup>6</sup>, без «многоэтажных» дробей).

Распространенной ошибкой начинающих программистов является запись инструкций присваивания в виде:

```
c = 2a
```

<sup>4</sup> Также см. дополнение.

<sup>5</sup> Их называют также «операторами».

<sup>6</sup> Такую запись называют «линейной».

(правильный вариант:  $c = 2 * a$ ; вариант инструкции  $c = a2$  возможен, если  $a2$  – имя переменной, значение которой задано).

При определении порядка действий используется приоритет (старшинство) операций. Они выполняются в следующем порядке:

- действия в скобках;
- возведение в степень ( $**$ ), справа налево (!);
- умножение ( $*$ ) и деление ( $/$ ), слева направо;
- сложение и вычитание, слева направо.

Таким образом, умножение и деление имеют одинаковый приоритет, более высокий, чем сложение и вычитание. Поэтому в последнем приведенном примере ( $sred = (a + b) / 2$ ) запись без скобок привела бы к тому, что сначала выполнялось бы деление значения  $b$  на 2, а затем сложение этого частного и значения переменной  $a$ .

Над величинами целого типа, кроме операций сложения, вычитания, умножения, деления и возведения в степень, можно выполнять также еще две операции:

- 1) определение целой части частного от деления одного целого числа на другое – знак операции « $//$ »;
- 2) определение остатка от деления одного целого числа на другое – знак операции « $\%$ ».

Пример программы:

```
a = 10
b = 3
z = a//b
m = a % b
print(z, m)
```

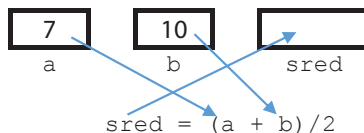
Результат ее выполнения:

3 1

Как работает инструкция присваивания? Объясним на примере:

```
sred = (a + b) / 2
```

Транслятор (см. главу 1) «посмотрит», что «лежит» в ячейках  $a$  и  $b$ , сложит соответствующие значения и сумму разделит на 2. Результат «положится» в ячейку  $sred$ :





Это означает, что в программе может быть запись:

```
x = x + 10
```

невозможная в математике. Ее смысл в том, что нужно к имеющемуся в данный момент выполнения программы значению переменной *x* прибавить 10, а результат присвоить той же переменной.

В Python разрешено множественное присваивание. Запись

```
a, b, c = 7, 2, -5
```

равносильна инструкциям

```
a = 7
b = 2
c = -5
```

а запись `a = b = 0` равносильна паре инструкций

```
b = 0
a = b
```

Часто используют сокращенную запись арифметических операций:

Сокращенная запись	Полная запись
<code>a += b</code>	<code>a = a + b</code>
<code>a -= b</code>	<code>a = a - b</code>
<code>a *= b</code>	<code>a = a * b</code>
<code>a /= b</code>	<code>a = a /b</code>

Рассмотрим задачу: «Даны два целых числа. Найти их сумму».

Ясно, что в программе следует использовать три переменные.

Пусть их имена *a*, *b*, *sum*.

Программа решения задачи:

```
#Ввод исходных данных
print('Задайте первое число')
a = input()
print('Задайте второе число')
b = input()
#Расчет суммы
sum = a + b
#Вывод ответа на экран
print('Сумма этих чисел равна', sum)
```

Однако, после того как мы запустим программу и введем какие-то числа, допустим, 15 и 22, мы увидим странный ответ: 1522. Дело в

том, что при нажатии клавиши на клавиатуре (в том числе клавиш с цифрами) в компьютер поступает код клавиши, то есть код соответствующего символа. И входные данные воспринимаются инструкцией `input()` именно как последовательность символов. Поэтому в приведенной программе переменные `a` и `b` будут заданы не как числа, а как цепочки символов, при сложении этих цепочек (с помощью оператора «+») программа просто объединяет их – приписывает вторую цепочку в конец первой (см. главу 11).

Чтобы исправить эту ошибку, нужно преобразовать символьную строку, которая получена при вводе, в целое число. Это делается с помощью функции `int()` (от англ. *integer* – целый):

```
#Ввод исходных данных
print('Задайте первое число')
a = int(input())
print('Задайте второе число')
b = int(input())
...
```

Итак, после того как мы преобразовали введенные значения в формат целых чисел, программа работает правильно – складывает два числа, введенных с клавиатуры.

Если надо ввести в программу вещественное число, то при использовании инструкции `input()` необходимо записать функцию `float()`:

```
print('Задайте вещественное число')
z = float(input())
```

Как уже отмечалось, целесообразно включить в инструкцию `input()` подсказку:

```
a = int(input('Введите целое число '))
z = float(input('Задайте вещественное число '))
```

Мы рассмотрели особенности присваивания значения величинам числового типа.

Если нужно присвоить значение переменной типа `str`, то в правой части инструкции указывается строка символов в кавычках (одинарных или двойных), имя переменной величины типа `str` или имя метода для работы со строками. Примеры:

```
club = 'Спартак'

moi_club = club

im = 'Dima'
im2 = im.upper()  #О методе upper() см. главу 3
```

Можно также указать несколько таких значений, между которыми записывается знак «+»:

```
Kto = im + ' ' + fam  
  
all = 'Я - болельщик команды' + ' ' + club
```

Другие операции<sup>7</sup> над строками выполнять нельзя (помните, что перечень операций определяется типом величины?).

Инструкция присваивания значений величинам типа `bool` будет рассмотрена в главе 5.

Обсудим одну интересную задачу программирования: «Даны значения двух переменных величин `a` и `b`. Произвести обмен их значений».

Кажущее очевидным решение в виде:

```
a = b  
b = a
```

или

```
b = a  
a = b
```

требуемого результата не даст (убедитесь в этом!). Как же быть? А так, как происходит обмен содержимого двух чашек, в одной из которых находится молоко, а в другой – чай. Нужна третья чашка!<sup>8</sup> То есть в нашей задаче для решения требуется третья, вспомогательная переменная. С ее использованием обмен может быть проведен следующим образом:

```
c = a    #Запоминаем значение величины a  
a = b    #Величине a присваиваем значение величины b  
b = c    #Величине b присваиваем 'старое' значение величины a
```

или

```
c = b  
b = a  
a = c
```

Так задача решается во всех языках программирования. В программе на Python имеется также возможность провести обмен оригинальным<sup>9</sup> способом:

<sup>7</sup> Возможна также операция «умножения» величины строкового типа на целое число, которая используется редко (см. главу 11).

<sup>8</sup> Можно, конечно, использовать и две дополнительные чашки, но это уже, так сказать, «слишком» ☺.

<sup>9</sup> Далее в книге будут приведены и другие оригинальные методы и функции Python. Но наряду с ними будут обсуждаться способы решения соответствующих задач без использования возможностей Python. Это полезно, потому что они заставляют думать, как решить соответствующую задачу, и раскрывают суть того, как ее решает система программирования. Кроме того, эти способы решения могут понадобиться вам, когда вы будете программировать на других языках.



$a, b = b, a$

Здесь использовано множественное присваивание (см. выше).

## Контрольные вопросы

1. Какие величины в программе называют «переменными»?
2. Чем характеризуется каждая переменная?
3. Каковы правила присвоения имен переменным?
4. Почему желательно переменным давать «говорящие» имена?
5. Какие типы переменных вы знаете?
6. Что определяет тип переменной?
7. Какие значения может иметь переменная логического типа?
8. Как можно смоделировать хранение значения переменной в памяти компьютера?
9. Как обратиться к значению (использовать значение) переменной величины в программе?
10. С помощью какой инструкции можно ввести в программу значение переменной в ходе ее выполнения?
11. В чем заключается особенность ввода в программу в ходе ее выполнения числовых значений переменных?
12. Почему желательно выводить на экран подсказку перед вводом данных?
13. С помощью какой инструкции можно изменить (или задать впервые) в программе значение переменной? Как она оформляется в общем случае? Как она оформляется применительно к переменным числового типа? К переменным типа `str`?
14. Что такое приоритет операций? Зачем он нужен? Перечислите арифметические операции в порядке уменьшения приоритета.
15. В каком порядке выполняются операции, если они имеют одинаковый приоритет?
16. Зачем в инструкции присваивания используются скобки?
17. Чем отличаются операции, знаки которых `</>`, `< // >` и `<%>`?

## Задания

1. Запишите в одну строку по правилам языка Python следующие арифметические выражения:

а)  $\frac{-1}{x^2}$ ;

е)  $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$ ;