ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Зачем изучать программирование?

Программирование развивает креативность, логическое мышление, а также навыки поиска и устранения ошибок. Программист может создавать что-то из ничего, пользуясь логикой для составления понятных компьютеру программных конструкций, а если что-то пойдет не так, он отыщет ошибку и исправит проблему. Писать программы — занятие увлекательное и временами непростое, однако полученный опыт пригодится и в школе, и дома (даже если ваша профессия не будет связана с компьютерами).

Ну и, по меньшей мере, программирование — отличный способ скоротать время, когда за окном непогода.

Почему именно Python?

Python — простой в изучении язык программирования, он особенно хорош для начинающих. В отличие от многих других языков, Python-код легко читается, а интерактивная оболочка позволяет вводить программы и сразу же получать результат. Помимо простой структуры языка и интерактивной оболочки, в Python есть инструменты, заметно ускоряющие обучение и позволяющие создавать несложные анимации для своих видеоигр. Один из таких инструментов — специально созданный для обучения модуль `turtle`, который имитирует «черепашью графику» (в 1960-х годах она использовалась в языке Logo). Другой инструмент — модуль `tkinter` для работы с графической библиотекой `Tk`, позволяющей создавать программы с продвинутой графикой и анимацией.

Как изучать программирование?

Как правило, если вы встречаетесь с чем-то впервые, лучше начинать с основ, поэтому читайте книгу с самого начала, поборов искушение сразу перепрыгнуть в середину или конец. Никто не в силах сыграть симфонию, впервые взяв в руки инструмент. Начинающие пилоты не поднимаются в небо, не изучив приборы управления. Гимнасты не могут (как правило) сделать сальто назад с первой попытки. Если вы перейдете к последним главам раньше времени, вы не только плохо усвоите базовые понятия, но и сами эти главы покажутся вам куда сложнее, чем они есть на самом деле.

Читая книгу, запускайте каждый из примеров кода. В конце большинства глав есть упражнения, которые помогут укрепить знания. Если

что-то покажется вам непонятным или чересчур сложным, советую действовать так:

1. Разбейте задачу на составные части. Постарайтесь сперва понять, что делает небольшой фрагмент кода (фокусируйтесь на кусочках программы, не пытайтесь с ходу разобраться, как она устроена целиком).

2. Если это не помогает, иногда проблему лучше всего отложить, чтобы вернуться к ней на другой день. Этот способ хорош для многих жизненных ситуаций, и особенно при изучении программирования.

Для кого эта книга

Эта книга — для всех, кто интересуется программированием, будь это ребенок или взрослый, которому программирование в новинку. Если вы хотите не просто пользоваться чужими разработками, а создавать свое, «Python для детей» — хороший способ приступить к делу.

Изучив основы программирования, вы узнаете, как создавать собственные игры. Вам предстоит разработать две игры, научившись определять столкновения, использовать события и применять разные способы анимации.

Большинство примеров в этой книге рассчитаны на программирование в среде IDLE, которая идет в комплекте с Python. IDLE поддерживает подсветку синтаксиса, копирование и вставку текста, а также возможность сохранения и загрузки вашего кода. То есть IDLE одновременно и интерактивная среда для экспериментов, и что-то вроде текстового редактора. Хоть для запуска примеров достаточно стандартной консоли и обычного редактора текстов, подсветка синтаксиса и дружественный интерфейс IDLE облегчат вашу задачу, поэтому мы обязательно разберемся, как настроить и использовать эту среду.

Что вас ждет?

Вот краткое описание материала каждой из глав.
Глава 1 — введение в программирование и инструкции по установке Python.
Глава 2 — знакомство с простыми вычислениями и с переменными.
Глава 3 — описание некоторых основных типов данных (таких как строки, списки, кортежи).
Глава 4 — знакомство с модулем `turtle`. От основ программирования мы перейдем к перемещению черепашки (она похожа на стрелочку) по экрану.
Глава 5 — описание логических условий и конструкции `if`.

Глава 6 — циклы `for` и `while`.
Глава 7 — введение в создание и использование функций.
Глава 8 — введение в классы и объекты. На этом этапе мы освоим достаточно базовых возможностей языка, чтобы использовать приемы программирования игр.
Глава 9 — обзор большинства встроенных функций Python.
Глава 10 — обзор нескольких модулей, которые идут в комплекте с Python.
Глава 11 — снова о модуле `turtle` и рисовании более сложных фигур.
Глава 12 — модуль `tkinter` и создание продвинутой графики.
Главы 13 и 14 — пишем нашу первую игру, «Прыг-скок!», используя знания, полученные в предыдущих главах.
Главы от 15 до 18 — создаем вторую игру, «Человечек спешит к выходу». При вводе кода из глав, посвященных играм, вы можете допустить ошибки. Если найти их самостоятельно не получится, скачайте код игры с сайта этой книги (*python-for-kids.com/* или *mann-ivanov-ferber.ru*) и сравните с ним вашу программу.
Послесловие — краткий обзор модуля PyGame и некоторых других популярных языков программирования.
Приложение — подробное описание ключевых слов Python.
Глоссарий — определения терминов из области программирования, которые встречаются в данной книге.

Повеселитесь!

Изучая эту книгу, помните, что программирование может быть очень увлекательным. Воспринимайте его не как работу, а как способ создания веселых игр и программ, которыми можно поделиться с другими людьми. Изучение программирования отлично тренирует ум, и результаты могут быть впечатляющими. Но главное — что бы вы ни делали, не забывайте веселиться!

ЧАСТЬ I

Учимся программировать

1

НЕ ВСЕ ЗМЕИ ПРЕСМЫКАЮТСЯ

Компьютерная программа — это набор инструкций, следуя которым компьютер выполняет различные действия. Программу не найти среди деталей компьютера: проводов, микросхем, карт памяти, жестких дисков и тому подобного. Ее невозможно увидеть, однако выполняется она с помощью аппаратуры. Компьютерная программа (или просто *программа*) состоит из последовательности команд, указывающих оборудованию, что и как делать. Совокупность работающих на компьютере программ называют *программным обеспечением*.

Практически любое из электронных устройств, которыми мы пользуемся, не будет работать или станет гораздо менее полезным, если лишить его программного обеспечения. Программы управляют не только компьютерами, но и мобильными телефонами, игровыми приставками, автомобильными GPS-навигаторами. Среди не столь очевидных примеров — жидкокристаллические телевизоры, DVD-плееры, микроволновые печи и некоторые модели холодильников. Даже двигатели автомобилей, светофоры и уличные фонари, электронные рекламные панели и лифты в наши дни работают благодаря программам.

Программы чем-то похожи на мысли. Если бы у нас не было мыслей, мы, наверное, сидели бы на полу, ничего не делая. Мысль встать с пола — это инструкция, или команда, которая говорит нашему телу, что нужно подняться. Так же и программы говорят компьютеру, как ему действовать.

Научившись программировать, вы сможете делать множество полезных вещей. Вряд ли вы будете создавать программы для автомобилей, светофоров или холодильников (во всяком случае, это требует специальной подготовки), однако вы сможете разрабатывать веб-страницы,

видеоигры и даже писать программы, помогающие делать домашние задания.

Немного о языке

Как и люди, компьютеры «говорят» на разных языках, только языки эти — компьютерные. *Компьютерный язык* служит для того, чтобы переговариваться с компьютером, используя команды, понятные и компьютеру, и человеку.

Некоторые языки программирования названы в честь людей (например, Ада и Паскаль), другие названия являются простыми акронимами, то есть аббревиатурой (к примеру, BASIC — от англ. Beginner's All-purpose Symbolic Instruction Code, универсальный код символических инструкций для начинающих), и уж совсем немногие языки названы в честь телевизионных шоу — как Python. О да, язык программирования Python (произносится «Пайтон», с ударением на первый слог, хотя имейте в виду, что в России многие называют язык просто «питон») получил свое имя благодаря телешоу «Летающий цирк Монти Пайтона», так что змея питон здесь вовсе ни при чем.

Python — *букв.* питон

«Летающий цирк Монти Пайтона» — британское комедийное телешоу, впервые вышедшее на экраны в 1970 году. Хотя съемки «летающего цирка» давно прекращены, у него множество поклонников по всему миру. Среди комедийных скетчей этого шоу есть, например, зарисовки «Министерство глупых походок», «Рыбошлепский танец» и «Сырная лавка» (в которой не продают сыр).

Благодаря некоторым особенностям Python отлично подходит для новичков. Главное — на нем можно писать простые и эффективные программы, не тратя на это много времени. В Python используется меньше сложных специальных символов, чем в большинстве других языков, так что программы на нем легко читаются. (Однако не думайте, что в программах на Python нет особых символов, просто они используются реже, чем во многих других языках.)

Установка Python

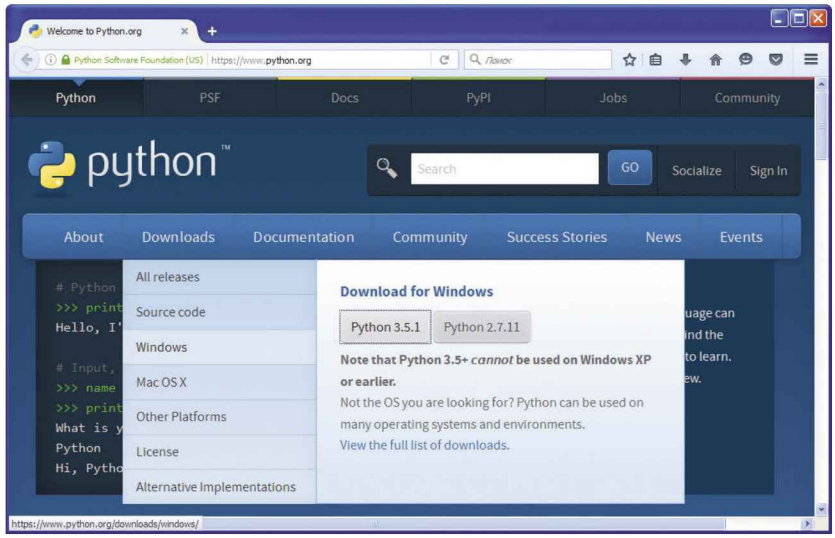
IDLE (от Integrated DeveLopment Environment) — интегрированная среда разработки

Установить Python на компьютер совсем несложно. Сейчас мы шаг за шагом разберем, как устанавливать его на системы Windows 7, Mac OS X и Ubuntu. Также мы создадим на рабочем столе ярлык для IDLE — среды разработки Python-программ. Если Python уже установлен на вашем компьютере, можете сразу переходить к разделу «Когда Python установлен» на стр. 20.

Установка Python в системе Windows 7

Чтобы установить Python в системе Microsoft Windows 7, откройте веб-браузер, введите адрес *http://www.python.org/* и скачайте последнюю версию программы-установщика Python 3 для Windows (для этого зайдите в меню *Downloads* и выберите *Windows*).

Download — скачать



Неважно, какую конкретно версию Python вы скачаете. Главное, чтобы ее номер начинался с цифры 3.

После того как установщик скачается, дважды кликните мышкой по его значку и установите Python, следуя инструкциям программы:

1. Выберите **Install for All Users** и нажмите **Next**.
2. Не меняйте указанный адрес установки, но запомните его (например, *C:\Python31 или C:\Python32*). Нажмите **Next**.
3. Ничего не меняйте в разделе установщика *Customize Python*, просто нажмите **Next**.

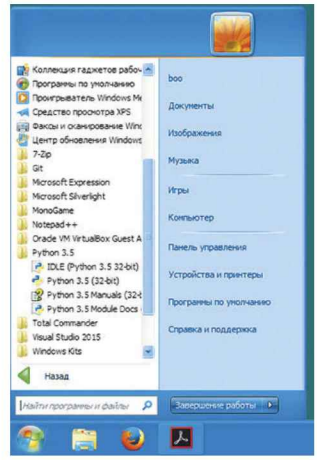
Install for All Users — установить для всех пользователей

Next — далее

Customize Python — настроить Python

После окончания установки в меню *Start* (*Пуск*) должен появиться раздел *Python 3*.

Start — пуск



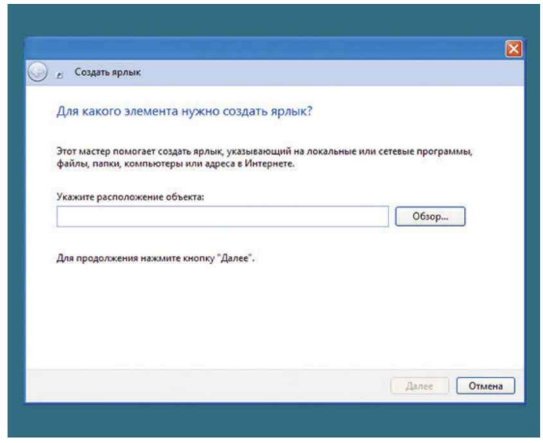
Теперь добавьте ярлык Python 3 на рабочий стол:

1. Кликните по рабочему столу правой кнопкой мышки и выберите из появившегося меню **New ▸ Shortcut (Создать ▸ Ярлык).**

2. Введите в поле с пометкой **Type the location of the item** (Укажите расположение объекта) следущую строку (каталог в начале этой строки должен соответствовать каталогу установки, который я просил вас запомнить):

```
c:\Python32\Lib\idlelib\idle.pyw -n
```

Диалоговое окно должно выглядеть так:



3. Нажмите **Next (Далее)**, чтобы перейти к следующему диалогу.

4. Укажите имя *IDLE* и нажмите **Finish (Готово)**, чтобы создать ярлык.

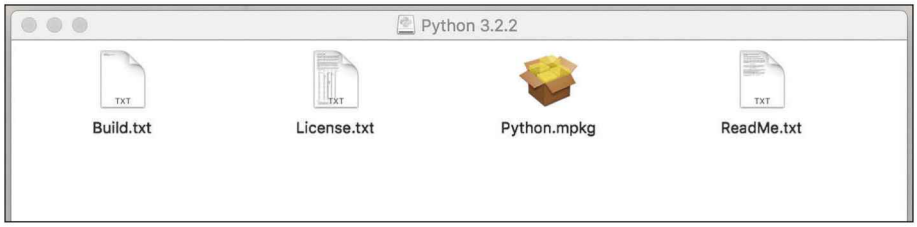
Теперь переходите к разделу «Когда Python установлен» на стр. 20 и начинайте знакомство с Python.

Установка Python в системе MAC OS X

Если у вас Mac, Python должен быть уже установлен в системе, однако скорее всего это одна из старых версий языка. Откройте веб-браузер, перейдите по адресу *http://www.python.org/getit/* и скачайте последнюю версию инсталлятора для Mac OS X.

Вам нужно выбрать инсталлятор в зависимости от вашей версии Mac OS X (чтобы узнать версию, кликните по значку с яблоком в верхнем меню и выберите пункт **About this Mac**).

После того как файл скачается, дважды клините по нему. Должно появиться окошко с содержимым файла.



В этом окне дважды кликните по значку *Python.mpkg* и следуйте инструкциям. Перед установкой система попросит вас ввести пароль администратора (не знаете пароль? Спросите у родителей).

Теперь добавьте на рабочий стол скрипт для запуска среды разработки IDLE:

1. Кликните по значку **Spotlight** — увеличительному стеклышку в правом верхнем углу экрана.

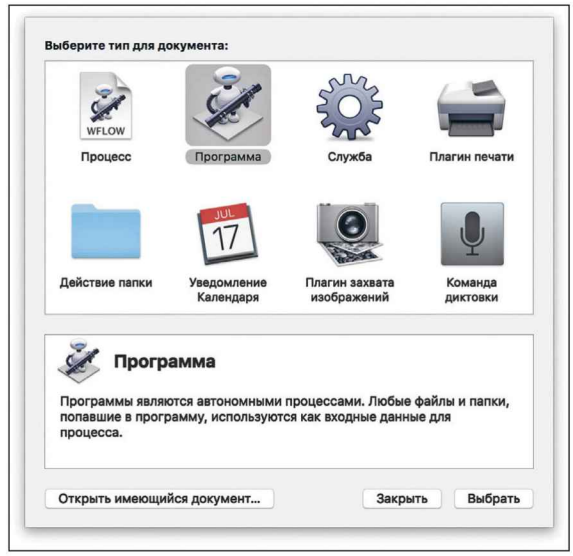
Spotlight — *букв.* прожектор

2. В появившемся поле введите *Automator*.

3. Кликните по приложению со значком в виде робота, когда оно появится в меню (либо в секции *Top Hit*, либо в секции *Applications*).

Application — приложение

4. После того как Automator запустится, выберите шаблон **Application** (Программа).

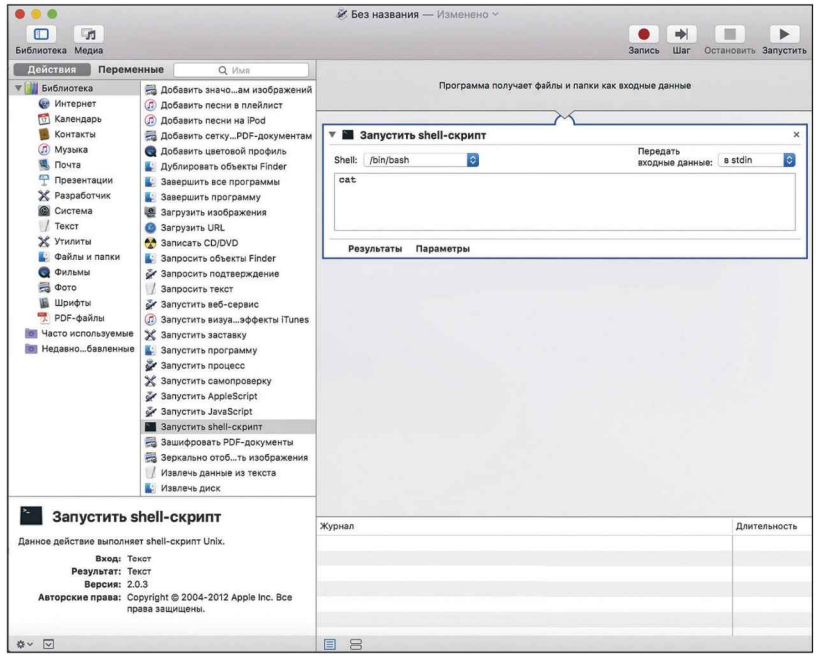


Choose — выбрать

5. Нажмите кнопку **Choose** (Выбрать).

Run — запустить
Shell Script — скрипт оболочки

6. Отыщите в списке действий пункт **Run Shell Script** (Запустить shell-скрипт) и перетащите его на пустую панель справа. Результат будет выглядеть примерно так:



7. В поле ввода вы увидите слово *cat*. Замените его такой строкой:

```
open -a "/Applications/Python 3.2/IDLE.app" —args -n
```

8. Выберите **File ▸ Save** (Файл ▸ Сохранить) и укажите в качестве имени *IDLE*.

9. В диалоге *Where* выберите **Desktop** (Рабочий стол) и нажмите **Save** (Сохранить).

Теперь переходите к разделу «Когда Python установлен» (на стр. 20) и начинайте знакомство с Python.

Установка Python в системе Ubuntu

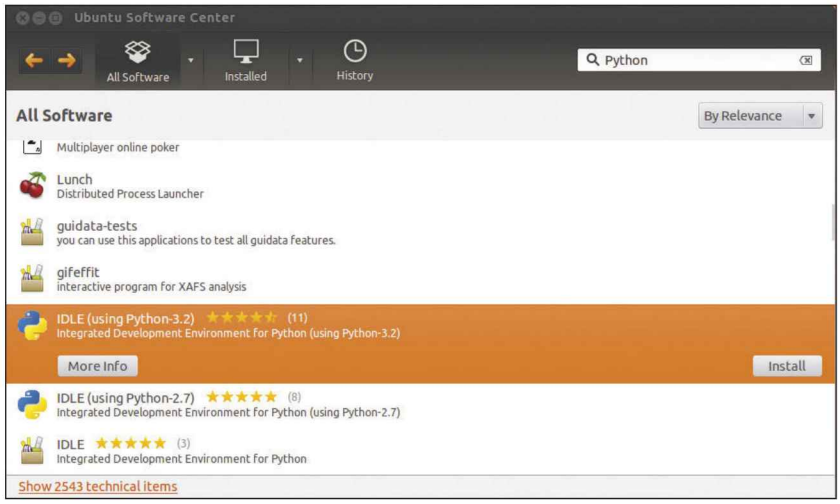
Python уже входит в Ubuntu Linux, но это может быть старая версия языка. Для установки Python 3 выполните следующие шаги:

1. В сайдбаре кликните по кнопке центра приложений Ubuntu (значок с оранжевой сумкой. Если вы его не видите, кликните по значку меню Dash и введите в строке поиска *Software*).

Software — программное обеспечение

2. Запустив центр приложений, введите *Python* в строке поиска (расположенной в правом верхнем углу окна).

3. В появившемся списке приложений выберите последнюю версию IDLE, например *IDLE (using Python 3.2)*:



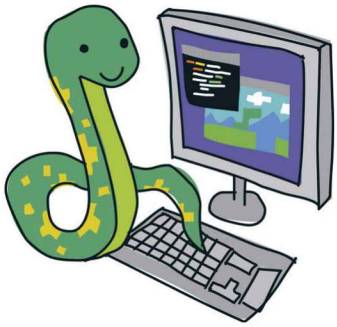
4. Нажмите **Install**.

Authenticate — авторизация

5. Введите пароль администратора и нажмите **Authenticate** (не знаете пароль? Спросите у родителей).

В некоторых версиях Ubuntu в списке будет лишь строка Python (v3.2) — без слова IDLE. Это тоже подходит.

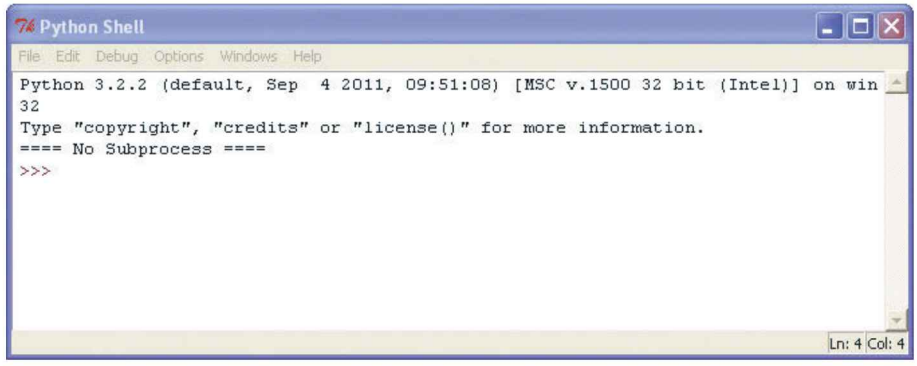
Итак, у вас установлена последняя версия Python. Давайте скорее посмотрим, что же это такое.



Когда Python установлен

Если вы пользуетесь Windows или Mac OS X, к этому моменту на вашем рабочем столе должен находиться значок с надписью *IDLE*. Если же вы используете Ubuntu, в меню *Applications* должен появиться раздел *Programming*, а в нем приложение *IDLE (using Python 3.2)* (или более поздняя версия).

Дважды кликните по значку или выберите приложение из меню. Должно появиться такое окно:



Это командная оболочка Python, которая входит в интегрированную среду разработки, а три знака «больше» (>>>) называются *приглашением*.

После приглашения можно вводить различные команды. Что ж, давайте приступим:

```
>>> print("Привет, мир")
```

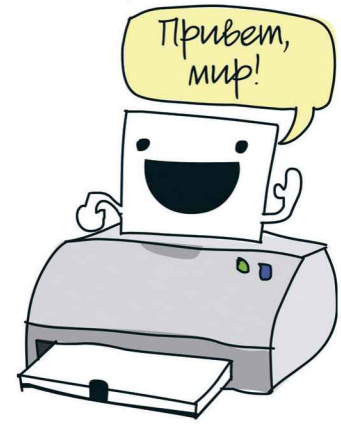
Не забудьте про двойные кавычки (" "). Закончив вводить эту строку, нажмите клавишу **Enter**. Если вы ввели команду без ошибок, на экране должно появиться:

```
>>> print("Привет, мир")
Привет, мир
>>>
```

Приглашение возникнет снова. Это значит, что оболочка Python готова к выполнению дальнейших команд.

Поздравляю! Вы только что создали первую программу на языке Python! Слово `print` относится к разновидности команд, которые называются *функциями*, и эта конкретная функция выводит на экран все, что указано после нее в двойных кавычках. То есть вы дали компьютеру команду напечатать слова «Привет, мир» и эта команда понятна и вам, и компьютеру.

Print — печать



Сохранение Python-программ

От программ было бы мало толку, если бы их каждый раз приходилось писать заново. Конечно, если программа совсем короткая, это несложно. Однако большие программы могут состоять из миллионов строк кода. Чтобы распечатать весь код такой программы, например редактора документов, потребуется не меньше 100 000 листов бумаги. Представьте, каково нести такую груду листов домой!

К счастью, тексты программ можно сохранять на диск. Чтобы сохранить новую программу, запустите IDLE и выберите в меню **File ▸ New File**. Откроется пустое окно со словом *Untitled* в заголовке. Введите в этом новом окне такой код:

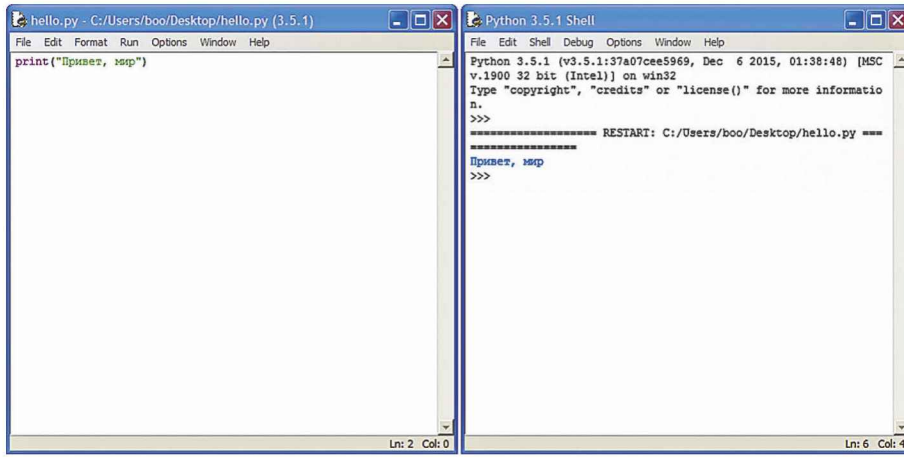
Untitled — без названия

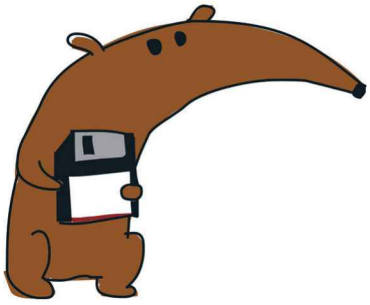
```
print("Привет, мир")
```

Теперь выберите в меню **File ▸ Save**. Введите в ответ на запрос имени файла *hello.py* и сохраните файл на рабочий стол. Теперь выберите **Run ▸ Run Module**. Сохраненная программа должна запуститься.

Run module — запустить модуль

Если вы закроете окно оболочки, оставив открытым окно с заголовком *hello.py,* и выберете из меню **Run ▸ Run Module**, окно оболочки появится снова и ваша программа запустится. Чтобы открыть оболочку Python без запуска программы, выберите **Run ▸ Python Shell**.





После сохранения и запуска этой программы вы обнаружите на рабочем столе новый значок с названием *hello.py*. Если дважды кликнуть по нему мышкой, на экране появится черное окно и тут же исчезнет. Что произошло?

Это было консольное окно Python (что-то вроде командной оболочки), где наша программа запустилась, напечатала слова «Привет, мир» и тут же завершила работу. Это окно практически невозможно рассмотреть, прежде чем оно закроется:

В IDLE для открытия окна оболочки, сохранения файла и запуска программы можно использовать не только команды меню, но и специальные сочетания клавиш:

- В системах Windows и Ubuntu используйте **CTRL-N** для создания нового окна, **CTRL-S** — для сохранения отредактированного файла и **F5** — для запуска программы.

- В системе Mac OS X используйте **⌘-N** для создания нового окна и **⌘-S** для сохранения файла, а для запуска программы нажмите клавишу **FN** и, удерживая ее, нажмите **F5**.

Что мы узнали

В этой главе мы создали программу «Привет, мир». По традиции с нее принято начинать изучение программирования. В следующей главе мы воспользуемся оболочкой Python для решения задач посложнее.

2

ВЫЧИСЛЕНИЯ И ПЕРЕМЕННЫЕ

Итак, вы установили Python и знаете, как запускать его командную оболочку, а значит, пора использовать его по назначению. Мы начнем с простых математических расчетов, а затем перейдем к важной части языка — переменным. *Переменные* — это удобный способ хранения данных в программе, и они пригодятся нам для решения самых разных задач.

Вычисления в Python

Если нужно перемножить два числа, к примеру узнать, сколько будет 8 × 3,57, мы обычно пользуемся калькулятором либо берем ручку и умножаем в столбик на листе бумаги. А что если использовать для подсчетов оболочку Python? Давайте попробуем.

Запустите оболочку, дважды кликнув по значку IDLE на рабочем столе, либо, если у вас система Ubuntu, кликнув по значку IDLE в меню **Applications**. Затем после значка >>> введите выражение и нажмите Enter:

```
>>> 8 * 3.57
28.56
```

Обратите внимание, что при записи числа 3,57 используется не запятая, а точка. Кроме того, в Python числа перемножаются с помощью звездочки (*), а не знака умножения (×).

Теперь рассмотрим более полезную задачу.

Представьте, что вы рыли яму и случайно нашли кошелек с 20 золотыми монетами. На следующий день вы тихонько залезли в подвал, где

стоит изобретение вашего дедушки — работающий на паровом ходу механизм для копирования предметов, и, на ваше счастье, в него удалось запихнуть все 20 монет. Раздался свист, потом щелчок, и устройство выдало еще 10 новеньких монеток.

Сколько монет вы накопите, если будете проделывать эту операцию каждый день в течение года? На бумаге эти расчеты выглядят примерно так:

$10 \times 365 = 3650$
$20 + 3650 = 3670$

Что ж, ничего сложного, осталось лишь выяснить, как посчитать то же в оболочке Python. Первым делом умножаем 10 монет на 365 дней, получится 3650. Затем добавим 20 монет, которые были изначально, и выйдет 3670.

```
>>> 10 * 365
3650
>>> 20 + 3650
3670
```

Но что если о вашем богатстве узнает пронырливая ворона? Предположим, она будет каждую неделю залетать в окно и красть по три монетки.

Сколько у вас будет монет через год? В оболочке Python эти расчеты будут выглядеть так:

```
>>> 3 * 52
156
>>> 3670 - 156
3514
```

Сперва умножаем 3 монеты на 52 недели в году, получаем 156. Затем вычитаем это значение из общего количества монет. Выходит, через год у вас останется 3514 монет.

Получилась очень простая программа. Изучая эту книгу дальше, вы узнаете, как писать более сложные и полезные программы.

Операторы в Python

В оболочке Python можно умножать, складывать, вычитать и делить числа, а также совершать некоторые другие операции, о которых мы узнаем позже. Символы, с помощью

которых выполняются математические действия в языке Python, называются *операторами*. Основные математические операторы перечислены в таблице 2.1.

Символ	Операция
+	Сложение
-	Вычитание
*	Умножение
/	Деление

Таблица 2.1. Основные операторы в Python

Прямой слеш (/) обозначает деление, этот символ похож на линию между числителем и знаменателем дроби. Например, у вас 100 пиратов и 20 больших бочек, и вы хотите рассчитать, сколько пиратов можно спрятать в каждой бочке. Для этого следует разделить 100 пиратов на 20 бочек, введя в оболочке 100 / 20. И запомните — прямым слешем называют черту, верх которой наклонен вправо.

Порядок выполнения операций

Операции — это любые действия, которые совершаются с помощью операторов. Математические операции выполняются по очереди в зависимости от их приоритета (если не задать другую очередность с помощью скобок). Умножение и деление имеют более высокий приоритет, чем сложение и вычитание, и это значит, что они будут выполняться первыми. Иначе говоря, при вычислении математического выражения Python сначала умножит и разделит числа, а затем перейдет к сложению и вычитанию.

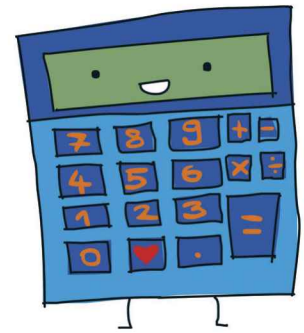
Например, в этом выражении сперва будут перемножены числа 30 и 20, а затем к их произведению будет прибавлено число 5.

```
>>> 5 + 30 * 20
605
```

По сути это выражение означает «умножить 30 на 20 и прибавить к результату 5». Получается 605. Однако мы можем изменить порядок операций, заключив первые два числа в скобки. Вот так:

```
>>> (5 + 30) * 20
700
```

В результате получилось 700, а не 605, поскольку Python выполняет операции в скобках прежде, чем операции вне скобок. Другими словами, это выражение означает «прибавить 5 к 30 и умножить результат на 20».



Скобки могут быть *вложенными*, то есть внутри скобок могут стоять еще одни скобки:

```
>>> ((5 + 30) * 20) / 10
70.0
```

В этом примере Python сперва вычислит выражение во внутренних скобках, затем во внешних и в самом конце выполнит стоящую за скобками операцию деления.

Иначе говоря, это выражение означает «прибавить 5 к 30, затем умножить результат на 20, потом разделить результат на 10». Вот что при этом происходит:

- сложение 5 и 30 дает 35;
- умножение 35 на 20 дает 700;
- деление 700 на 10 дает окончательный результат — 70.

Если бы мы не использовали скобки, результат вышел бы другим:

```
>>> 5 + 30 * 20 / 10
65.0
```

В этом случае сперва 30 умножается на 20 (получается 600), затем 600 делится на 10 (выходит 60) и, наконец, к 60 прибавляется 5, что дает в итоге 65.

Запомните, что умножение и деление всегда выполняются прежде, чем сложение и вычитание, если не менять порядок вычислений с помощью скобок.

Переменные как ярлыки для данных

В программировании слово *переменная* обозначает именованное место для хранения данных, например чисел, текста, списков с числами или символами и так далее. Также переменную можно рассматривать как ярлык, которым помечены некие данные.

Например, чтобы создать переменную с именем `fred`, нужно указать имя, поставить знак «равно» (=) и ввести соответствующие данные. Давайте создадим переменную `fred` (Фред), указав, что ей соответствует значение 100 (однако из этого не следует, что другая переменная не может иметь такое же значение):

```
>>> fred = 100
```

Чтобы напечатать значение нашей переменной, введите в оболочке Python команду `print` и следом за ней — имя переменной в скобках. Вот так:

```
>>> print(fred)
100
```

Можно изменить значение переменной `fred` — сделать так, чтобы ей соответствовали другие данные. Например, вот как заменить значение `fred` числом 200:

```
>>> fred = 200
>>> print(fred)
200
```

В первой строке говорится, что переменной `fred` теперь соответствует число 200. Во второй строке мы запрашиваем значение `fred`, чтобы убедиться, что оно поменялось. Последней строкой Python печатает ответ.

Можно использовать несколько переменных для одного и того же значения:

```
>>> fred = 200
>>> john = fred
>>> print(john)
200
```

В этом примере знак «равно» между именами `john` (Джон) и `fred` говорит о том, что переменной `john` соответствует значение переменной `fred`.

Конечно, `fred` — не самое удачное имя переменной, поскольку оно не поясняет, для чего эта переменная используется. Лучше назовем переменную не `fred`, а, допустим, `number_of_coins` (количество монет):

```
>>> number_of_coins = 200
>>> print(number_of_coins)
200
```

Number of coins — количество монет

Теперь понятно, что речь идет о двухстах монетах.

Имена переменных могут состоять из латинских букв, цифр и знака подчеркивания (_), однако начинаться с цифры они не могут. В остальном допустимо использовать любые имена, которые могут состоять как из отдельных букв (например, `a`), так и из целых предложений (пробелы в именах недопустимы, но слова можно разделять знаками подчеркивания). Для небольших программ часто удобны короткие имена, но в целом желательно, чтобы имя переменной отражало смысл, который вы вкладываете в ее использование.

Теперь вы знаете, как создавать переменные. Давайте посмотрим, что с ними можно делать.

Использование переменных

Помните, как мы вычисляли, сколько монет накопится за год, если каждый день создавать новые монеты с помощью изобретения вашего дедушки? Итак, вот на чем мы остановились:

```
>>> 20 + 10 * 365
3670
>>> 3 * 52
156
>>> 3670 - 156
3514
```

Все это можно записать одной строкой кода:

```
>>> 20 + 10 * 365 - 3 * 52
3514
```

А что если заменить в этом выражении числа переменными? Введите:

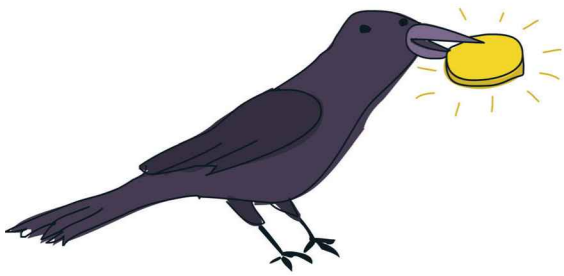
```
>>> found_coins = 20
>>> magic_coins = 10
>>> stolen_coins = 3
```

Found coins — найденные монеты
Magic coins — волшебные монеты
Stolen coins — украденные монеты

Мы создали три переменные: `found_coins`, `magic_coins` и `stolen_coins`.

Теперь можно записать наше выражение так:

```
>>> found_coins + magic_coins * 365 – stolen_coins * 52
3514
```

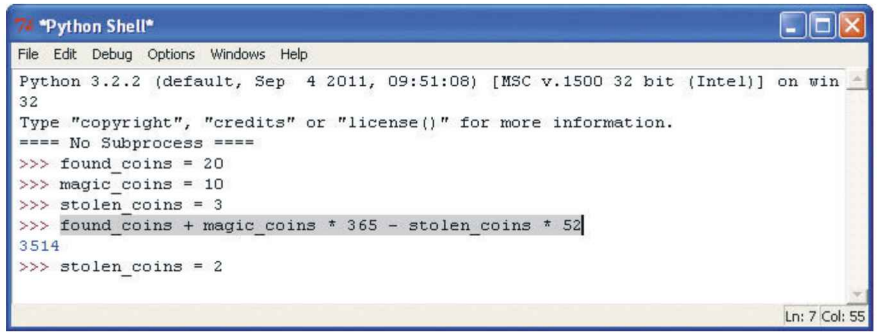


Как видите, результат остался прежним. А зачем все это? Ради особой магии переменных. Представьте, что вы поставили у окошка пугало и поэтому осторожная ворона крадет по две, а не по три монеты. Если использовать в расчетах переменную, достаточно поменять ее значение на другое число, чтобы оно использовалось везде, где эта переменная фигурирует. То есть мы можем поменять значение `stolen_coins`, введя:

```
>>> stolen_coins = 2
```

А затем скопировать и снова вставить наше выражение, чтобы пересчитать ответ. Вот так:

1. Выделите текст, который нужно скопировать. Для этого кликните по нему и, не отпуская кнопку мышки, перетащите область выделения от начала строки до ее конца, как показано на рисунке:

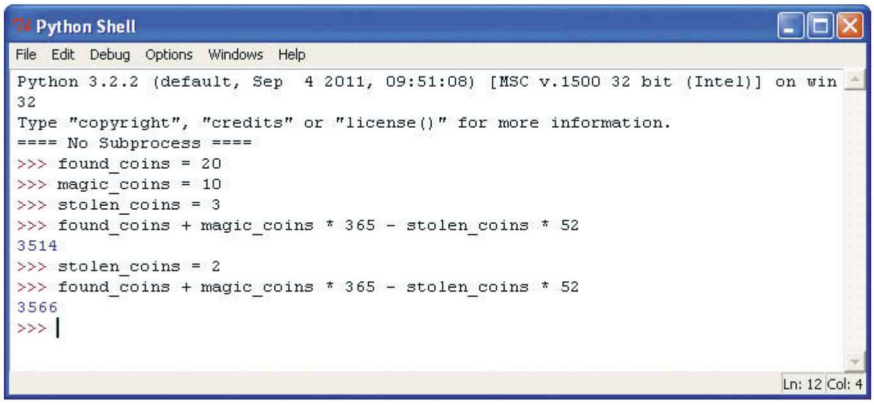


2. Нажав и удерживая клавишу **Ctrl** (или, если вы используете Mac OS, клавишу ⌘), нажмите **C**, чтобы скопировать выделенный текст (далее я буду называть это действие **Ctrl-C**).

3. Кликните по строке с приглашением (следующей после `stolen_coins = 2`).

4. Нажав и удерживая **Ctrl**, нажмите **V**, чтобы вставить скопированный ранее текст (далее я буду называть это действие **Ctrl-V**).

5. Нажмите **Enter**, чтобы заново вычислить результат:



Не правда ли, так гораздо проще, чем заново вводить все выражение?

Можете поменять значения других переменных, а потом опять скопировать (**Ctrl-C**) и вставить (**Ctrl-V**) выражение, чтобы увидеть результат изменений. Предположим, вы обнаружили, что если стукнуть по дедушкиному изобретению, из него вылетает на 3 монеты больше. Поступая так каждый раз, через год вы получите 4661 монету:

```
>>> magic_coins = 13
>>> found_coins + magic_coins * 365 – stolen_coins * 52
4661
```

Конечно, в таком простом выражении от переменных *не очень* много толку. До настоящей пользы мы еще не дошли. Пока просто запомните, что переменные — это способ присваивать именам значения для их дальнейшего использования.

Что мы узнали

В этой главе мы разобрались, как составлять простые выражения с помощью операторов языка Python, использовать скобки для изменения порядка операций (очередности, в которой Python вычисляет части выражений), научились создавать переменные, присваивать им значения и использовать в расчетах.

3

СТРОКИ, СПИСКИ, КОРТЕЖИ И СЛОВАРИ

В предыдущей главе мы выполняли простые расчеты, а также познакомились с переменными. В этой главе мы научимся работать с еще несколькими конструкциями языка Python: строками, списками, кортежами и словарями. Строки пригодятся для вывода текста (например, сообщений «Старт!» или «Игра окончена» в компьютерной игре), а в списках, кортежах и словарях можно хранить наборы значений.

Строки

Фрагменты текста в программировании обычно называют *строками*. Можно сказать, что строка — это последовательность символов. Из всех букв, пробелов, цифр и других печатных знаков в этой книге можно составить строку, и из вашего имени или адреса тоже. По сути, первая программа на Python, которую мы создали в главе 1, уже включала в себя строку «Привет, мир».

Текст в кавычках нужно вводить без переносов. Здесь его пришлось перенести, поскольку он не вместился в ширину страницы, и это показано значком ↵. Встретив такой значок, вводите текст без переноса.

Создание строк

Чтобы создать строку, нужно ввести текст в кавычках — так Python отличает строки от чисел и других типов данных. Например, возьмем переменную `fred` из главы 2 и присвоим ей строковое значение:

```
fred = "Почему у горилл большие ноздри? Потому что у них ↵
толстые пальцы!"
```

Теперь напечатаем значение `fred`, воспользовавшись командой `print(fred)`:

```
>>> print(fred)
Почему у горилл большие ноздри? Потому что у них толстые пальцы!
```

Строку можно записать и в одинарных кавычках:

```
>>> fred = 'Что это: розовое и пушистое? Розовый пушистик!'
>>> print(fred)
Что это: розовое и пушистое? Розовый пушистик!
```

Однако если вы попытаетесь перенести текст, который начинается с одинарной (') или двойной (") кавычки на новую строку или поставить в начале текста кавычку одного типа, а в конце — другого, Python выдаст сообщение об ошибке. Например, введите:

```
>>> fred = "Что едят на полдник динозавры?
```

И вот что получится:

```
SyntaxError: EOL while scanning string literal
```

Python выдал сообщение о синтаксической ошибке, потому что, вопреки правилам, мы не завершили строку одинарной или двойной кавычкой.

Синтаксическая ошибка — это неверное расположение слов в предложении или — в нашем случае — слов и символов в программе. Сообщение `SyntaxError` означает, что вы ввели данные не в том порядке, который ожидает Python, или не ввели те данные, которые он от вас ждал. Здесь Python, дойдя до конца строки, не обнаружил закрывающую кавычку и выдал ошибку.

Syntax Error — синтаксическая ошибка

Если нужно ввести текст, занимающий несколько строк, поставьте в начале и в конце три одинарные кавычки, а когда понадобится сделать перенос, нажимайте **Enter**. Вот так:

```
>>> fred = '''Что едят на полдник динозавры?
ТиРекс-кекс!'''
```

Теперь напечатаем значение переменной `fred`:

```
>>> print(fred)
Что едят на полдник динозавры?
ТиРекс-кекс!
```

Проблемы со строками

Теперь посмотрите на следующую строку. Попытка ее ввести приведет к сообщению об ошибке:

Silly string — глупая строка
Invalid syntax — недопустимый синтаксис

```
>>> silly_string = '"Тут что-то не так, не будь я д'Артаньян" — ↵
подумал он.'
SyntaxError: invalid syntax
```

Мы попытались создать (и присвоить переменной `silly_string`) строку в одинарных кавычках, в которой есть двойные кавычки и еще одна одинарная в слове `"д'Артаньян"`. Ну и беспорядок!

Не забывайте, что Python не обладает человеческим разумом, поэтому видит только строку, за которой следуют лишние символы. Когда Python встречает кавычку (одинарную или двойную) в начале строки, он считает, что такая же кавычка должна стоять и в конце. Поскольку в этом примере строка начинается с одинарной кавычки, Python воспринимает одинарную кавычку после буквы «д» как конец строки. Напечатав сообщение об ошибке, Python подсвечивает проблемное место в коде (это слово «Артаньян» после закрывающей кавычки):