Содержание

ГЛАВЫ

ПРИЛОЖЕНИЯ

Изучите программирование на языке Python, создав простую игру в кости

Python — удобный язык для людей всех возрастов, с опытом программирования или без него.

Python [1] — это универсальный язык программирования, который может применяться для создания приложений, трехмерной графики, видеоигр и даже веб-сайтов. Это отличный первый язык программирования, так как его легко изучать, и он проще, чем такие сложные языки, как C, C++ или Java. Однако, несмотря на кажущуюся простоту, Python — достаточно мощный и надежный язык для создания сложных приложений, и он используется практически во всех отраслях, где применяются компьютеры. Это делает Python удобным языком для людей всех возрастов, как с опытом программирования, так и без него.

Установка Python

Перед изучением языка Python может потребоваться его установка.

Linux: Если вы используете Linux, то Python уже включен в комплект поставки, но необходимо убедиться, что у вас установлен именно Python 3. Чтобы проверить, какая версия установлена, откройте окно терминала и введите:

```
python –version
```

Если при этом выяснится, что у вас установлена версия 2 или вообще нет никакой версии, попробуйте повторить команду, указав вместо нее Python 3:

```
python3 --version
```

Если эта команда не найдена, то необходимо установить Python 3 из менеджера пакетов или программного центра. Какой именно менеджер пакетов используется в вашем дистрибутиве Linux, зависит от дистрибутива. Наиболее распространенными являются **dnf** в Fedora и **apt** в Ubuntu. Например, в Fedora вы набираете следующее:

```
sudo dnf install python3
```

MacOS: Если вы работаете на Mac, следуйте тем же инструкциям, которые использовали для Linux, чтобы проверить, установлен ли у вас Python 3. В macOS

нет встроенного менеджера пакетов, поэтому если Python 3 не найден, установите его с сайта python.org/downloads/mac-osx [2]. Даже если в вашей версии macOS уже установлен Python 2, вам все же следует изучить Python 3.

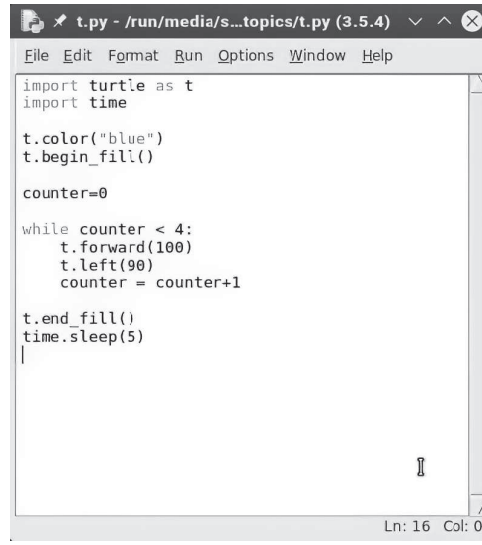
Windows: В настоящее время Microsoft Windows не поставляется с Python. Установите его с сайта python.org/downloads/windows [3]. В мастере установки обязательно выберите **Add Python to PATH**. Инструкции по установке Python на Windows можно найти в моей статье How to Install Python on Windows [4].

Запуск среды разработки

Для написания программ на языке Python необходим только текстовый редактор, но гораздо удобней работать в интегрированной среде разработки (IDE). IDE объединяет в себе текстовый редактор и некоторые дружественные и полезные функции Python. IDLE 3 и PyCharm (Community Edition) — два варианта из многих [5], на которых стоит остановить свой выбор.

IDLE 3

Python поставляется с базовой IDE, называемой IDLE.



IDLE

В ней есть подсветка ключевых слов для обнаружения опечаток, подсказки при вызове, автозавершение и кнопка **Run** для быстрой и удобной проверки кода. Чтобы запустить среду, выполните следующие действия.

- В Linux или macOS запустите окно терминала и введите **idle3**.
- В операционной системе Windows запустите Python 3 из меню Пуск.
  - Если Python не отображается в меню “Пуск”, запустите командную строку Windows, набрав **cmd** в меню “Пуск”, и введите **C:\Windows\py.exe**.

- Если это не помогло, попробуйте переустановить Python. В мастере установки обязательно выберите опцию **Add Python to PATH**. Подробные инструкции приведены на сайте docs.python.org/3/using/windows.html [6].
- Если это все равно не помогает, просто используйте Linux. Эта операционная система бесплатна, и если вы сохраните файлы Python на USB-накопителе, вам даже не придется устанавливать ее, чтобы использовать.

PyCharm Community Edition

PyCharm (Community Edition) IDE [7] — отличная IDE для языка Python с открытым исходным кодом. В ней есть подсветка ключевых слов, помогающая обнаружить опечатки, автоподстановка кавычек и скобок, позволяющая избежать синтаксических ошибок, номера строк (полезно при отладке), маркеры отступов и кнопка **Run** для быстрого и удобного тестирования кода.

Для того чтобы использовать PyCharm, необходимо сделать следующее.

1. Установите IDE PyCharm (Community Edition). В Linux ее проще всего установить с помощью Flatpak [8]. В качестве альтернативы можно загрузить [9] нужную версию инсталлятора с сайта PyCharm и установить ее вручную [10]. На macOS или Windows скачайте и запустите программу установки с сайта PyCharm [11].
2. Запустите PyCharm.
3. Создайте новый проект.

Как указать Python, что надо сделать

Ключевые слова сообщают Python, что ему нужно сделать. В новом файле проекта введите в IDE следующее:

```
print(“Hello world.”)
```

- Если вы используете IDLE, зайдите в меню Run и выберите опцию Run module.
- Если вы используете PyCharm, нажмите кнопку Run File в левой панели кнопок.

opensource.com

Ключевое слово **print** указывает Python на необходимость вывести на печать любой текст, который вы передадите ему в круглых скобках и кавычках.

Однако это не впечатляет. Основное ядро языка Python имеет доступ только к базовым ключевым словам, таким как **print**, **help**, основные математические функции и т. д.

Для загрузки большего количества ключевых слов можно использовать ключевое слово **import.**

Turtle — это очень занимательный модуль. Введите этот код в свой файл (заменив старый код), а затем запустите его:

```
import turtle

turtle.begin_fill()
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.end_fill()
```

Посмотрите, какие фигуры можно нарисовать с помощью модуля turtle.

Чтобы очистить область рисования в модуле turtle, используйте ключевое слово **turtle.clear()**. Как вы думаете, что делает ключевое слово **turtle.color(“blue”)**?

Улучшаем программу

Для получения аналогичных результатов можно попробовать более сложный код. Вместо того чтобы вручную кодировать каждую линию и каждый поворот, можно воспользоваться **циклом while**, попросив Python сделать это четыре раза подряд: нарисовать линию и затем повернуть. Python может отслеживать, сколько раз он выполнил эти действия, с помощью *переменной* **counter**. Вскоре вы узнаете больше о переменных, а пока попробуйте определить, как взаимодействуют переменная **counter** и **цикл while**:

```
import turtle as t
import time

t.color(“blue”)
t.begin_fill()

counter = 0

while counter < 4:
    t.forward(100)
    t.left(90)
    counter = counter + 1
```

```
t.end_fill()
time.sleep(2)
```

После того как вы запустили свою программу, пришло время изучить еще более совершенный модуль.

Изучение языка Python через создание игры

Чтобы лучше узнать, как работает Python, и подготовиться к более сложному программированию с использованием графики, давайте остановимся на игровой логике. В этом учебном пособии мы также немного узнаем о том, как устроены компьютерные программы, создав текстовую игру, в которой компьютер и игрок бросают виртуальный кубик, и побеждает тот, у кого выпало большее число.

Планирование игры

Прежде чем писать код, необходимо подумать о том, что именно вы собираетесь написать. Многие программисты, прежде чем приступить к написанию кода, создают простую документацию [12], чтобы иметь цель, к которой нужно стремиться. Вот как могла бы выглядеть программа для игры в кости, если бы вместе с игрой поставлялась документация.

1. Запустите игру в кости и нажмите Return или Enter для броска.
2. Результаты выводятся на экран.
3. Вам будет предложено выполнить повторный бросок или выйти из программы.

Это простая игра, но в документации подробно расписано то, что необходимо сделать. Например, в ней говорится, что для написания этой игры необходимы следующие компоненты.

- Игрок: для участия в игре необходим человек.
- AI: компьютер тоже должен бросать кубик, иначе игроку не у кого выигрывать или некому проигрывать.
- Случайное число: обычный шестигранный кубик выдает рандомное число от 1 до 6.
- Оператор сравнения: простая математика позволяет сравнить одно число с другим, чтобы определить, какое из них больше.
- Сообщение о победе или поражении.
- Запрос на повторную игру или выход из игры.

Создание начальной версии игры в кости

В начальную версию игры, как правило, не включают все ее возможности, а начинают программирование с основной логики. Сначала введем несколько определений.

Переменная — это значение, которое может изменяться. Переменные очень часто используются в Python. Всякий раз, когда вам нужно, чтобы ваша программа что-то “запомнила”, вы используете переменную. Фактически почти вся инфор-

мация, с которой работает код, хранится в переменных. Например, в математическом уравнении **x** + **5** = **20** переменной является *x*, поскольку буква *x* — это место для хранения какого-то значения.

Целое число — это число, которое может быть как положительным, так и отрицательным. Например, 1 и -1 — целые числа. Так же как и 14, 21 и даже 10 947.

Переменные в Python легко создавать и легко с ними работать. В этой начальной версии игры в кости используются две переменные: **player** и **ai**.

Создайте новый проект, назовите его **dice_alpha** и наберите в нем следующий код:

```
import random

player = random.randint(1,6)
ai = random.randint(1,6)

if player > ai :
    print("Вы победили")              # обратите внимание на отступы
else:
    print("Вы проиграли")
```

Запустите игру, чтобы убедиться в ее работоспособности.

Эта базовая версия игры в кости работает довольно хорошо. Она решает основные задачи игры, но пока еще не очень похожа на игру как таковую. Игрок никогда не узнает, что выпало ему, а что — компьютеру. А игра заканчивается, даже в том случае, если игрок хотел бы сыграть еще раз.

Это характерно для первых версий программ (так называемых альфа-версий). Теперь, когда вы уверены, что сможете запрограммировать основную часть игры (бросок кубика), пришло время дополнить программу.

Улучшение игры

В этой второй версии (называемой бета-версией) вашей игры несколько улучшений позволят сделать ее более похожей на настоящую игру.

1. Опишите результаты

Вместо того чтобы просто сообщать игрокам, выиграли они или проиграли, гораздо интереснее, если они будут знать, какое именно число у них выпало. Попробуйте внести эти изменения в свой код:

```
player = random.randint(1,6)
print("У вас выпало " + player)

ai = random.randint(1,6)
print("У компьютера выпало " + ai)
```

Если вы запустите игру сейчас, то произойдет сбой, поскольку Python думает, что вы пытаетесь выполнить математические вычисления. Он думает, что вы хотите сложить буквы “У вас выпало” и любое число, которое в данный момент хранится в переменной player.

Вы должны указать Python, чтобы он обращался с числами в переменных player и ai именно как со строками в предложении, а не как с числами в математическом уравнении.

Внесите следующие изменения в свой код:

```
player = random.randint(1,6)
print("У вас выпало " + str(player))

ai = random.randint(1,6)
print("У компьютера выпало " + str(ai))
```

Запустите игру, чтобы увидеть результат.

2. Немного замедлим

Компьютеры быстры. Люди иногда тоже могут быть быстрыми, но в играх лучше создать интригу. Вы можете использовать функцию **time** для замедления игры в какие-то напряженные моменты.

```
import random
import time

player = random.randint(1,6)
print("У вас выпало " + str(player))

ai = random.randint(1,6)
print("Компьютер делает бросок...")
time.sleep(2)
print("У компьютера выпало " + str(player))

if player > ai:
    print("Вы победили")    # обратите внимание на отступы
else:
    print("Вы проиграли")
```

Запустите игру, чтобы увидеть изменения.

3. Обнаружение закономерностей

Если вы достаточно поиграете в свою игру, то обнаружите, что, хотя внешне она работает правильно, на самом деле в ней есть ошибка: она не знает, что делать, если у игрока и компьютера выпадает одно и то же число.

Чтобы проверить, равно ли одно значение другому, Python использует ==. Это *два* знака равенства, а не один. Если используется только один знак равенства, Python думает, что вы пытаетесь создать новую переменную, но на самом деле вы пытаетесь выполнить математическую операцию сравнения.

Если вы хотите иметь более двух вариантов (кроме как выиграть или проиграть), вы можете использовать ключевое слово Python **elif**, что означает *else if*. Это позволяет проверять, истинно ли одно из *нескольких* значений, а не проверять, истинно ли *одно* значение.

Измените свой код следующим образом:

```
if player > ai:
    print("Вы победили")    # обратите внимание на отступы
elif player == ai:
    print("Ничья.")
else:
    print("Вы проиграли")
```

Запустите свою игру несколько раз, чтобы попытаться сравняться с компьютером.

Программирование финального релиза

Бета-версия вашей игры в кости функциональна и больше похожа на игру, чем альфа-версия. Для окончательного варианта создайте свою первую **функцию** Python.

Функция — это набор кода, к которому можно обращаться как к отдельной единице. Функции важны, поскольку большинство приложений содержат много кода, но не весь этот код должен выполняться одновременно. Функции позволяют запускать приложение и контролировать, что и когда происходит.

Измените свой код следующим образом:

```
import random
import time

def dice():
    player = random.randint(1,6)
    print("У вас выпало " + str(player))

    ai = random.randint(1,6)
    print("Компьютер делает бросок...")
    time.sleep(2)
    print("У компьютера выпало " + str(ai))

    if player > ai:
        print("Вы победили")          # обратите внимание на отступы
    else:
        print("Вы проиграли")

    print("Выход? Y/N")
    continue = input()

    if continue == "Y" or continue == "y":
        exit()
    elif continue == "N" or continue == "n":
        pass
    else:
        print("Ваш выбор непонятен. Сыграете еще раз?")
```

Эта версия игры спрашивает игрока, хочет ли он выйти из игры после ее окончания. Если игрок отвечает **Y** или **y**, вызывается функция **exit** языка Python, и игра завершается.

Важный момент заключается в том, что вы создали свою собственную функцию под названием **dice**. Функция **dice** запускается не сразу. На самом деле, если попробовать игру на этом этапе, то она не завершится, но и не запустится. Для того чтобы функция **dice** действительно выполняла какие-то действия, необходимо **вызвать ее** в коде.

Добавьте этот цикл в нижнюю часть существующего кода. Первые две строки приведены только для контекста и для того, чтобы подчеркнуть, где есть отступ, а где нет. Обращайте пристальное внимание на отступы.

```
    else:
        print("Ваш выбор непонятен. Сыграете еще раз?")

# главный цикл
while True:
    print("Нажмите кнопку ввода, чтобы бросить кубик.")
    roll = input()
    dice()
```

Первым выполняется блок кода **while True**. Поскольку **True** является истиной по определению, этот блок кода выполняется до тех пор, пока Python не прикажет ему выйти.

Блок кода **while True** представляет собой цикл. Сначала он предлагает пользователю начать игру, а затем вызывает вашу функцию **dice**. Так запускается игра. После того как функция **dice** отработала, цикл либо запускается снова, либо завершается, в зависимости от того, как игрок ответил на запрос.

Использование цикла для выполнения программы является наиболее распространенным способом кодирования приложений. Цикл гарантирует, что приложение остается в работе достаточно долго, чтобы пользователь компьютера мог использовать все функции приложения.

Следующие шаги

Теперь вы знаете основы программирования на языке Python. В следующей статье цикла мы расскажем о том, как написать видеоигру с помощью модуля PyGame [13], который имеет гораздо больше возможностей, чем модуль turtle, но он и гораздо сложнее.

Ссылки

[1] https://www.python.org/
[2] https://www.python.org/downloads/mac-osx/
[3] https://www.python.org/downloads/windows
[4] https://opensource.com/article/19/8/how-install-python-windows
[5] https://opensource.com/resources/python/ides
[6] https://docs.python.org/3/using/windows.html
[7] https://www.jetbrains.com/pycharm/download
[8] https://flathub.org/apps/details/com.jetbrains.PyCharm-Community
[9] https://www.jetbrains.com/pycharm/download/#section=linux
[10] https://opensource.com/article/18/1/how-install-apps-linux
[11] https://www.jetbrains.com/pycharm/download
[12] https://opensource.com/article/17/8/doc-drivendevelopment
[13] https://www.pygame.org/news

Создание игрового фреймворка на языке Python с помощью модуля Pygame

В первой главе этой книги мы познакомились с языком Python, создав простую игру с кубиками. Теперь пришло время создать свою собственную игру с нуля.

В первой главе я рассказал о том, как с помощью Python создать симулятор простой текстовой игры в кости. Вы также использовали модуль Turtle для создания простой графики. В этой статье мы начнем использовать модуль Pygame для создания игры с графикой.

Модуль Turtle входит в состав Python по умолчанию. Любой, у кого установлен Python, имеет и устанавливаемый автоматически вместе с ним модуль Turtle. Этого нельзя сказать о таком сложном модуле, как Pygame. Поскольку это специализированная библиотека кода, то Pygame необходимо устанавливать самостоятельно. Современная разработка на Python использует концепцию *виртуального окружения*, которая предоставляет коду Python собственное пространство для выполнения, а также помогает управлять библиотеками кода, которые использует приложение. Благодаря этому, когда вы передаете свое приложение Python другому пользователю для игры, вы точно знаете, что ему нужно установить, чтобы оно заработало.

Вы можете управлять виртуальным окружением Python вручную, а можете прибегнуть к помощи своей среды разработки. На данный момент всю эту работу можно поручить PyCharm. Если вы не используете PyCharm, прочитайте статью Ласло Кисса Коллара об управлении пакетами Python [1].

Начало работы с Pygame

Pygame — это библиотека, или *модуль языка Python*. Это набор кода, который позволяет не изобретать велосипед при создании каждой новой игры. Вы уже использовали модуль Turtle и можете представить, насколько сложным он мог бы быть, если бы вам пришлось писать код для создания пера, прежде чем рисовать им. Pygame предлагает аналогичные преимущества, но для видеоигр.

Видеоигре необходим сеттинг — мир, в котором она происходит. В Pygame существует два различных способа создания сеттинга:

- установка цвета фона;
- установка фонового изображения.

В любом случае фон — это только изображение или цвет. Персонажи видеоигры не могут взаимодействовать с предметами на заднем плане, поэтому не размещайте там ничего важного. Это просто декорации.

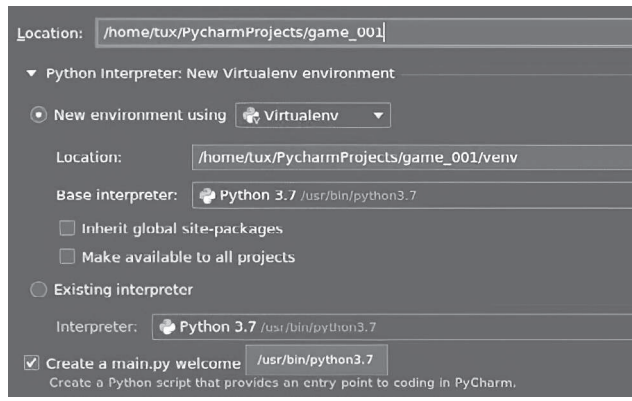
Настройка вашего первого скрипта Pygame

Чтобы начать новый проект на языке Python, обычно создают новую папку на компьютере и помещают в нее все игровые файлы. Очень важно, чтобы все файлы, необходимые для запуска игры, находились в папке проекта.

PyCharm (или любая другая используемая вами IDE) может сделать это за вас.

Чтобы создать новый проект в PyCharm, перейдите в меню **File** и выберите пункт **New Project**. В появившемся окне введите имя проекта (например, **game_001**.) Обратите внимание, что проект сохраняется в специальной папке **PycharmProjects** в домашнем каталоге. Таким образом, все файлы, необходимые игре, будут находиться в одном месте.

При создании нового проекта позвольте PyCharm создать новое окружение с помощью Virtualenv и примите все значения по умолчанию. Включите возможность создания файла **main.py** (и установки некоторых важных параметров по умолчанию).



opensource.com

После нажатия кнопки **Create** в окне PyCharm появляется новый проект. Проект состоит из виртуальной среды (каталог **venv,** указанный в левой колонке) и демонстрационного файла **main.py**.

Удалите все содержимое файла **main.py,** чтобы можно было ввести свой собственный код. Программа на языке Python начинается с указания типа файла, вашего имени и лицензии, которую вы хотите использовать. Используйте лицензию с открытым исходным кодом, чтобы ваши друзья могли улучшать вашу игру и делиться с вами своими изменениями:

```
#!/usr/bin/env python3
# by Seth Kenlon

# GPLv3
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License as
# published by the Free Software Foundation, either version 3 of the
# License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but
```

```
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
```

Затем укажите Python, какие модули вы хотите использовать. В этом примере кода не нужно вводить символ # или что-либо после него в каждой строке. Символ # в Python представляет собой *комментарий*, заметку в коде, оставленную для себя и других программистов.

```
import pygame    # загрузить ключевые слова pygame
import sys       # позволить python использовать вашу файловую систему
import os        # поможет python определить вашу ОС
```

Обратите внимание, что PyCharm не понимает, что такое Pygame, и подчеркивает это слово, чтобы отметить как ошибку. Это происходит потому, что Pygame, в отличие от sys и os, по умолчанию не включен в состав Python. Для успешного использования Pygame в коде необходимо включить его в каталог проекта. Чтобы сделать это, наведите курсор мыши на слово **pygame,** пока не появится всплывающее уведомление, объясняющее ошибку.

Щелкните на ссылке **Install package pygame** в окне оповещения и дождитесь, пока PyCharm установит Pygame в вашу виртуальную среду.

После его установки ошибка исчезнет.

Не имея такой среды разработки, как PyCharm, можно установить Pygame в виртуальную среду вручную с помощью команды pip.

Делим код на секции

Поскольку вы будете много работать с этим файлом программы, полезно создать внутри него функциональные разделы, чтобы знать, куда затем помещать отдельные фрагменты кода. Дадим каждому разделу свое название, отобразив его в строке комментария. Комментарии видны только при просмотре исходного текста программы. Итак, разделите вашу программу на четыре секции. Все комментарии Python проигнорирует, но это будут очень полезные отправные пункты, которые вы будете заполнять по мере изучения дальнейших уроков.

Я до сих пор использую такие отправные пункты при написании кода, поскольку это помогает мне лучше организовывать и планировать свою работу.

```
'''
Переменные
'''

# поместить сюда переменные

'''
Объекты
'''

# поместить сюда классы и функции Python
```