Библиотеки. Часть 1

План урока

1. Библиотеки как наследие
2. Репозиторий PyPI
3. Встроенные модули
4. Модуль random
5. Модуль wave

Аннотация

Python - это высокоуровневый язык программирования, объектно-ориентированный, модульный и подчеркнуто легкочитаемый, что делает его очень простым в изучении. Python широко применяется в образовательной сфере, для научных вычислений, больших данных и машинного обучения, в веб-разработке, графике, GUI, играх и других направлениях. В связи с огромной сферой применения, уже существуют бесчисленное количество библиотек, упрощающие программирование на этом языке без необходимости написания излишнего кода. На этом уроке мы начнем знакомство со стандартными библиотеками языка Python.

Библиотеки как наследие

У каждого языка программирования есть свои особенности — в том числе и у Python. Есть сферы деятельности, где он любим, а есть области, в которых он не очень силен и не пользуется популярностью.

Сначала поговорим про сильные стороны.

- На Python легко научиться программировать (именно поэтому вы его и изучаете).
- Благодаря строению языка и его динамической скриптовой природе, разрабатывать на Python можно очень быстро.

Принцип прост: **быстро изучить, быстро программировать**.

В результате Python заинтересовались люди, которым нужно писать программы, что-то автоматизировать или «склеивать» несколько взаимосвязанных программ в комплексы. Обычно эти люди не программисты по специальности. Это инженеры, преподаватели, математики, физики, биологи, лингвисты — все они часто применяют Python в своей практике.

Вторая причина популярности Python связана с тем, что он очень быстро стал интегрироваться с огромным количеством библиотек, написанных на других языках программирования. Что-то очень похожее было когда-то с языком программирования Perl и его платформой CPAN (http://www.cpan.org/).

Теперь пришел черед познакомиться с термином библиотека в контексте программирования.

Как говорит Википедия,

> **Библиоте́ка** (от англ. library) в программировании — сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО).

Давайте немного поясним. Правила структурного программирования гласят, что любая программа должна иметь структуру, а именно делиться на взаимодействующие компоненты (блоки): файлы, функции, классы. Это можно сравнить с разделением книг на главы, абзацы, слова — без такого деления трудно понять смысл. Этот принцип — первый важный этап для понимания сущности библиотеки.

Следующий этап — это возможность использовать часть кода одной программы из другой программы. Обычно это нужно для того, чтобы повторно использовать код: не писать одно и то же два раза, а также не переписывать код, написанный на другом языке программирования. Такое свойство часто называют модульностью.

Кстати, если бы все зарядки от сотовых телефонов были одинаковыми, вы бы могли насладиться модульностью в мире смартфонов. Был такой проект Ara (https://ru.wikipedia.org/wiki/Project_Ara) — модульный смартфон, в который можно было ставить дополнительную память, элементы питания, фотокамеру.

Самый простой контейнер для кода — функция. Часто используемые функции объединяются в библиотеки по своему типу. Например, функции по работе с видео-файлами, функции, отвечающие за соединение и получение информации из интернета и т. д.

Библиотеки могут объединяться в более крупные библиотеки — иногда в итоге получаются настоящие монстры, относящиеся к какой-то широкой области. Например, OpenCV — это библиотека для компьютерного зрения, Django — для веб-программирования, Scipy — для научных вычислений.

В Python библиотеки называются **модулями**.

Итак, современный язык программирования напоминает наборы конструкторов, которые совместимы друг с другом.

Если вы программируете на Python — представьте себя ребенком, попавшим на огромный склад с конструкторами, где можно взять любой конструктор и делать с ним что угодно.

Именно так. Очень многие библиотеки абсолютно свободны (на их использование нет никаких ограничений), бесплатны, а их исходные коды доступны.

Некоторые библиотеки, например, связанные с математическими вычислениями, очень старые и написаны ещё на Fortran. При этом они используются в современных проектах, поскольку созданы профессиональными математиками и инженерами, многократно проверены и оптимизированы.

В этом смысле **библиотеки** — такое же наследие человечества, как литература, музыка и архитектура.

Интересный факт: с точки зрения закона, программы являются литературными произведениями.

PyPI

PyPI (https://pypi.python.org/pypi) — это центральный репозиторий (хранилище) модулей для языка программирования Python. Он как PlayMarket для Android, AppStore для iPhone или CPAN для Perl.

Пройдите по ссылке. Вы увидите страницу, которая начинается со следующих слов:

> PyPI - the Python Package Index
>
> The Python Package Index is a repository of software for the Python programming language. There are currently 97430 packages here. To contact the PyPI admins, please use the Support (http://sourceforge.net/tracker/?group_id=66150&atid=513504) or Bug reports (https://github.com/pypa/pypi-legacy/issues) links.

Обратите внимание, что количество модулей превышает **125000**!

Наверное, можно в шутку говорить, что у Python на все случаи жизни есть нужная библиотека.

Допустим, вы хотите написать программу-бота для ВКонтакте, чтобы она делала за вас репосты, ставила лайки, переписывалась с друзьями, предлагала подружиться... Так вот, для этого тоже есть библиотека!

Как работать с PyPI, мы изучим на следующем уроке, а пока разберёмся со встроенными модулями.

Встроенные модули

Говорят, что Python поставляется «с батарейками в комплекте» — даже стандартной библиотеки, входящей в комплект поставки, уже достаточно для многих вещей.

Стандартной библиотеке посвящен целый раздел документации (https://docs.python.org/3/library/). Советуем вам хотя бы раз просмотреть его, чтобы примерно знать, какие вообще библиотеки бывают.

Модули в Python устроены по иерархическому принципу — как каталоги в файловой системе. Один модуль может быть вложен в другой, причем вложенность не ограничена (хотя на практике редко бывает больше 4).

Чтобы пользоваться функциями, объектами и классами из модуля, весь этот модуль или его часть нужно подключить к программе — **импортировать**.

Возникает вопрос: а почему бы не подключить все библиотеки сразу?

Можно, но это привело бы к нерациональному использованию оперативной памяти и очень долгой загрузке вашей программы. Поэтому есть правило: *не импортируйте то, чем не пользуетесь*.

За импорт в Python отвечает директива **import**.

Давайте посмотрим на примерах, как это происходит.

```
from math import pi # Возьмем число Пи из библиотеки math
```

Теперь вам доступна переменная pi. (В Python это значение приближенно равно 3.141592653589793)

Модуль, переменную, класс или функцию можно при импорте назвать своим именем — для этого служит ключевое слово **as**, например:

```
from math import pi as число_пи
print(число_пи)
```

Более того, поскольку в программе на языке Python в именах допустимы буквенные символы любых алфавитов, можно использовать даже греческие буквы (впрочем, это неудобно, если у вас кириллическо-латинская клавиатура).

```
from math import pi as π
print(π)
```

Если нужно импортировать что-то с большей степенью вложенности, то вам поможет символ ".", он выполняет ту же функцию, что и разные виды слэшей в путях до файлов.

```
# мне нужна функция urlopen из request,
# который находится внутри urllib
from urllib.request import urlopen
```

Мы можем импортировать всю библиотеку, но тогда для доступа к ее содержимому нужно снова использовать точку:

```
import math
print(math.pi)
```

Или несколько точек. В любом случае, аналогия с файловой структурой почти полная (объекты, функции и классы лежат в файлах, которые группируются в папки, которые тоже могут лежать в папках и т. д.).

```
import urllib
urllib.request.urlopen(...)
```

Значения после директивы `import` можно писать через запятую:

```
from math import sin, cos, tan
```

Значок "*" означает, что из библиотеки нужно импортировать всё, что доступно.

```
from math import *
```

Впрочем, так делать не рекомендуется, поскольку при таком подходе засоряется пространство имен.

```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'c
os', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'facto
rial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log1
0', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

```
help(math.sin)
```

```
Help on built-in function sin in module math:

sin(...)
    sin(x)

    Return the sine of x (measured in radians).
```

```
help(math.radians)
```

```
Help on built-in function radians in module math:

radians(...)
    radians(x)

    Convert angle x from degrees to radians.
```

```
sin(radians(30))
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-4-402ae401982b> in <module>()
----> 1 sin(radians(30))

NameError: name 'sin' is not defined
```

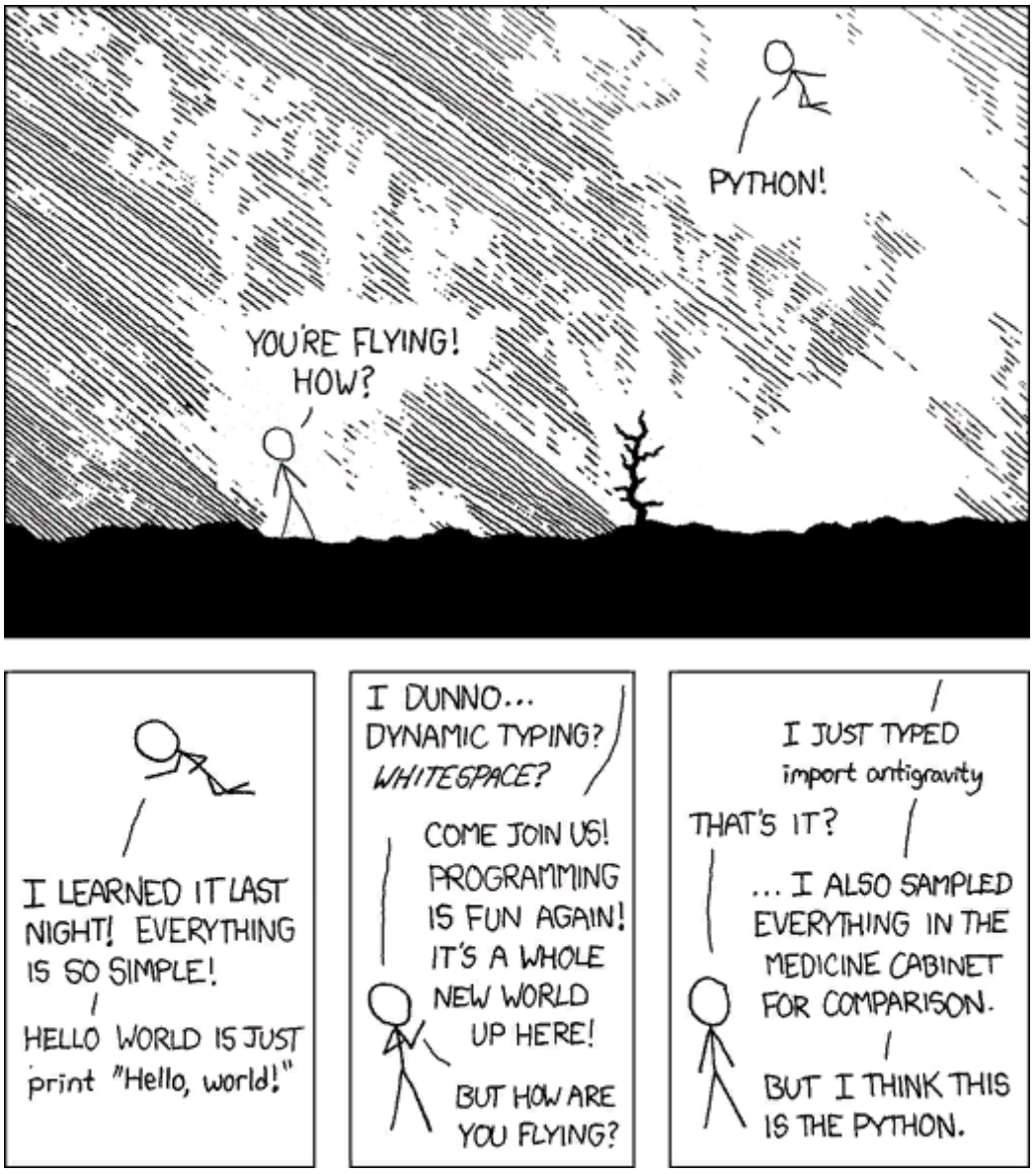
Обратите внимание, что часть имен начинается с символа _. Это служебные имена — мы их пока рассматривать не будем.

Говоря про встроенную библиотеку, нельзя не сказать о «пасхальных яйцах (https://ru.wikipedia.org/wiki/Пасхальное_яйцо_(виртуальное))» в Python. При импорте модуля `this` вы познакомитесь с **дзеном** (философией) Python.

```
import this
```

А импорт модуля с антигравитацией откроет в браузере комикс о том, что в Python действительно есть модули на все случаи жизни.

```
import antigravity
```



Давайте попробуем поработать с двумя интересными модулями: **`random`** и **`wave`**.

Модуль `random`

Этот модуль предназначен для работы с псевдослучайными последовательностями. Такие последовательности важны в математическом моделировании, в криптографии, а также в различных играх.

Давайте посмотрим структуру модуля.

```
import random
print(dir(random))
```

```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST',
'SystemRandom', 'TWOPI', '_BuiltinMethodType', '_MethodType', '_Sequence',
'_Set', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '_
_loader__', '__name__', '__package__', '__spec__', '_acos', '_bisect', '_c
eil', '_cos', '_e', '_exp', '_inst', '_itertools', '_log', '_pi', '_rando
m', '_sha512', '_sin', '_sqrt', '_test', '_test_generator', '_urandom', '_
warn', 'betavariate', 'choice', 'choices', 'expovariate', 'gammavariate',
'gauss', 'getrandbits', 'getstate', 'lognormvariate', 'normalvariate', 'pa
retovariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstat
e', 'shuffle', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariat
e']
```

Как видим, довольно много функций.

Одна из самых популярных — функция **`choice()`**. С её помощью можно выбрать один вариант из нескольких альтернатив, заданных в списке, кортеже, строке или любом другом итерируемом типе. Например, вот так можно моделировать подкидывание монетки:

```
from random import choice
choice((1, 2))
```

```
2
```

```
choice(["орел", "решка"])
```

```
'решка'
```

```
choice("ab")
```

```
'b'
```

А так — сымитировать несколько бросков игральных кубиков:

```
from random import choice
dashes = [1, 2, 3, 4, 5, 6]
for i in range(1, 10):
    print(choice(dashes), choice(dashes))
```

```
1 4
1 2
5 5
6 4
4 3
1 4
2 1
6 3
5 6
```

Если задать символы на сторонах кубика с использованием кодировки Юникод, всё будет ещё реалистичнее.

```
from random import choice

dashes = ['\u2680', '\u2681', '\u2682', '\u2683', '\u2684', '\u2685']
for i in range(1, 10):
    print(choice(dashes), choice(dashes))
```

```
⚃ ⚅
⚂ ⚄
⚀ ⚃
⚂ ⚀
⚄ ⚀
⚃ ⚅
⚀ ⚁
⚂ ⚄
⚂ ⚃
```

Можно сделать симуляцию магического шара с ответами (magic 8 ball (https://ru.wikipedia.org/wiki/Magic_8_ball)).

```
from random import choice

choices = [
    'It is certain (Бесспорно)',
    'It is decidedly so (Предрешено)',
    'Without a doubt (Никаких сомнений)',
    'Yes – definitely (Определённо да)',
    'You may rely on it (Можешь быть уверен в этом)',
    'As I see it, yes (Мне кажется – «да»)',
    'Most likely (Вероятнее всего)',
    'Outlook good (Хорошие перспективы)',
    'Signs point to yes (Знаки говорят – «да»)',
    'Yes (Да)',
    'Reply hazy, try again (Пока не ясно, попробуй снова)',
    'Ask again later (Спроси позже)',
    'Better not tell you now (Лучше не рассказывать)',
    'Cannot predict now (Сейчас нельзя предсказать)',
    'Concentrate and ask again (Соберись и спроси опять)',
    'Don’t count on it (Даже не думай)',
    'My reply is no (Мой ответ – «нет»)',
    'My sources say no (По моим данным – «нет»)',
    'Outlook not so good (Перспективы не очень хорошие)',
    'Very doubtful (Весьма сомнительно)',
]

for i in range(5):
    input("Ваш вопрос: ")
    print(choice(choices))
```

Вот как может выглядеть работа этой программы:

```
Ваш вопрос: Программировать хорошо?
Yes – definitely (Определённо да)
Ваш вопрос: Есть ли перспективы у Python?
You may rely on it (Можешь быть уверен в этом)
Ваш вопрос: Нет ли тайного заговора против школьников?
Signs point to yes (Знаки говорят – «да»)
Ваш вопрос: Может быть правительство?
Without a doubt (Никаких сомнений)
Ваш вопрос: Со мной будет что-то плохое из-за того, что я это узнал?
It is decidedly so (Предрешено)
```

Другая функция, расположенная в модуле **random** — как ни странно — `random()`. Она возвращает случайное **вещественное** число из интервала [0.0..1.0).

```
import random

for _ in range(5):
    print(random.random())
```

```
0.6208526162133347
0.08594424057466066
0.6098135340851979
0.35879884036106446
0.8947892532703161
```

Функция **randint(a, b)** возвращает случайное **целое** число из диапазона [a, b]. Обратите внимание, что интервал — закрытый.

Функция **randrange(a, b, [step])** является своего рода комбинацией `randint()` и `range()`. Она возвращает случайное число из диапазона range(a, b, [step]).

Ну и закончим рассмотрение библиотеки **random** функцией **shuffle()**. Она является антиподом сортировки: если **sort()** упорядочивает список, то **shuffle()** перемешивает его:

```
import random

data = list(range(10))
print('sorted:   ', data)
random.shuffle(data)
print('shiffled: ', data)
```

```
sorted:    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
shiffled:  [5, 1, 6, 2, 8, 0, 7, 3, 4, 9]
```

Модуль wave

Этот модуль предназначен для манипуляции с «сырыми», необработанными аудиоданными.

В ОС Windows такие данные хранятся в файлах с расширением **.wav**.

Сырые аудиоданные представляет собой зависимость амплитуды звукового сигнала от времени. Посмотрите на график этой зависимости. Вы часто видели его в фильмах, репортажах журналистов и т. д.

Вдоль оси абсцисс откладывается время, вдоль оси ординат — амплитуда (интенсивность, громкость) звукового сигнала.

Этот график строится по точкам, причём вместо пар `(t, y)` хранятся только значения `y`, а ось абсцисс задана частотой дискретизации (количеством отсчетов в секунду) — как правило, она составляет 44 100 Гц. Такой частоты достаточно, чтобы человек мог прослушать оцифрованный звук и не заметить его отличия от реального.

Получается, что представление звукового файла для программиста очень простое. Это список целых чисел (положительных и отрицательных) — значения амплитуды звукового сигнала.

В следующем примере мы рассмотрим основные этапы работы со звуковым файлом.

Наша программа будет разворачивать звуковой файл в обратную сторону, то есть проигрывать музыкальное произведение задом наперед.

```
import wave
import struct

source = wave.open("in.wav", mode="rb")
dest = wave.open("out.wav", mode="wb")

# установим параметры нового файла
dest.setparams(source.getparams())

# найдем количество фреймов
frames_count = source.getnframes()

data = struct.unpack("<" + str(frames_count) + "h", source.readframes(frames_count))

# собственно, основная строка программы - переворот списка
newdata = data[::-1]

newframes = struct.pack("<" + str(len(newdata)) + "h", *newdata)

# записываем содержимое в преобразованный файл.
dest.writeframes(newframes)
source.close()
dest.close()
```

Мы открываем два файла: исходный — **source** и формируемый — **dest**.

Одно значение амплитуды в терминах библиотеки **wave** называется **фреймом**.

Наибольшую трудность в нашем примере представляют строки с использованием встроенного модуля `struct`. Пока договоримся, что функции этого модуля могут на лету распаковывать и запаковывать данные разной природы.

Интересно, что палиндромы в аудиофайлах сохраняют свою «палиндромность» (например, «А роза упала на лапу Азора»).

Что ещё можно сделать?

- Если мы уберем, например, каждый второй фрейм, то ускорим произведение вдвое. При этом частота тоже вырастет в два раза («голос Чипа и Дейла»). Кстати, на телевидении и радио часто ускоряют видео и аудио на 5-10%: это незаметно для уха, но позволяет разместить больше рекламы в эфире.
- Если мы увеличим все фреймы в какое-то количество раз, то сделаем произведение громче, а если уменьшим — тише.
- Копируя каждый фрейм 2 раза, мы замедлим воспроизведение и понизим частоту.

Перед тем, как переходить к практике, запомните один из самых важных моментов:

Невозможно запомнить и выучить принципы работы всех функций даже из известных библиотек.

Поэтому, не забывайте о справочниках и документации (https://docs.python.org/3.6/library/index.html).