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 о сильных сторонах.

Сильные стороны Python

- 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 языком Python заинтересовались люди, которым нужно автоматизировать какие-то процессы в своей деятельности 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.

Теперь пришел черед познакомиться с термином «библиотека» в контексте программирования.

Как говорит Википедия,

Библиотека

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

Давайте немного поясним. Правила структурного программирования говорят, 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 в понимании сущности библиотеки.

Следующий этап заключается в возможности использовать часть кода одной программы из другой программы. Обычно это нужно, чтобы повторно использовать код: не писать одно и то же два раза. А еще — не переписывать код, написанный на другом языке программирования. Такое свойство часто называют модульностью.

Кстати, если бы все зарядки от электронных устройств были одинаковыми, вы бы могли насладиться модульностью в мире гаджетов. Был такой проект Ara — модульный смартфон, в который можно было ставить дополнительную память, элементы питания, фотокамеру.

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

Библиотеки могут объединяться в более крупные сборники, в итоге иногда 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 из них 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.

Иногда путают три понятия, связанные с ПО. Путаница получилась из-за того, что на английском free — это и беплатный, и свободный:

- Свободное ПО означает, что после покупки вы можете делать с ним всё, что хотите, включая дизассемблирование, использование в коммерческих проектах, написание на его основе своих программ;
- Бесплатное ПО;
- ПО с открытыми исходными кодами.

Возможны любые сочетания этих свойств.

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.

Репозиторий PyPI

PyPI — 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:

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community.

Обратите внимание: количество модулей уже превысило **300 000**!

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 разберемся со встроенными модулями.

Встроенные модули

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

Стандартной библиотеке посвящен целый раздел документации. Советуем вам хотя бы раз просмотреть его, чтобы примерно знать, какие вообще библиотеки бывают.

Модули в Python

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.

Важно!

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).

Ключевое слово as

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)
```

```
3.141592653589793
```

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)
```

```
3.141592653589793
```

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

```
# мне нужна функция urlopen из request,

# который находится внутри urllib

>>> from urllib.request import urlopen
```

Важно!

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

```
import math



print(math.pi)
```

```
3.141592653589793
```

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(...)
```

Особенности импорта модулей

Значения после директивы import можно писать через запятую:

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

Значок * означает, что из библиотеки нужно импортировать все, что доступно.

```
from math import *
```

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

Можно вспомнить об использовании двух полезных функций: **dir** (возвращает список со всем содержимым объекта, модуля и т. д.) и **help** (показывает справку об использовании данного объекта).

```
import math

print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
'acos',

 'acosh','asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil',
'copysign',

 'cos', 'cosh','degrees', 'e', 'erf', 'erfc', 'exp', 'expm1',
'fabs',

 'factorial', 'floor','fmod', 'frexp', 'fsum', 'gamma', 'hypot',
'isfinite',

 'isinf', 'isnan', 'ldexp','lgamma', 'log', 'log10', 'log1p',
'log2',

 'modf', 'pi', 'pow', 'radians', 'sin','sinh', 'sqrt', 'tan',
'tanh', 'trunc']
```

```
import math



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

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


sin(...)

    sin(x)


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

```
import math


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

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


radians(...)

    radians(x)


    Convert angle x from degrees to radians.
```

```
import math


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

```
0.49999999999999994
```

Обратите внимание: часть имен начинается с символа "_". Это служебные имена, мы их пока рассматривать не будем, да и программисты ими пользуются редко.

«Пасхальные яйца» в Python

Говоря про встроенную библиотеку, нельзя не сказать о «пасхальных яйцах» в Python. При импорте модуля `this` вы познакомитесь с дзеном Python.

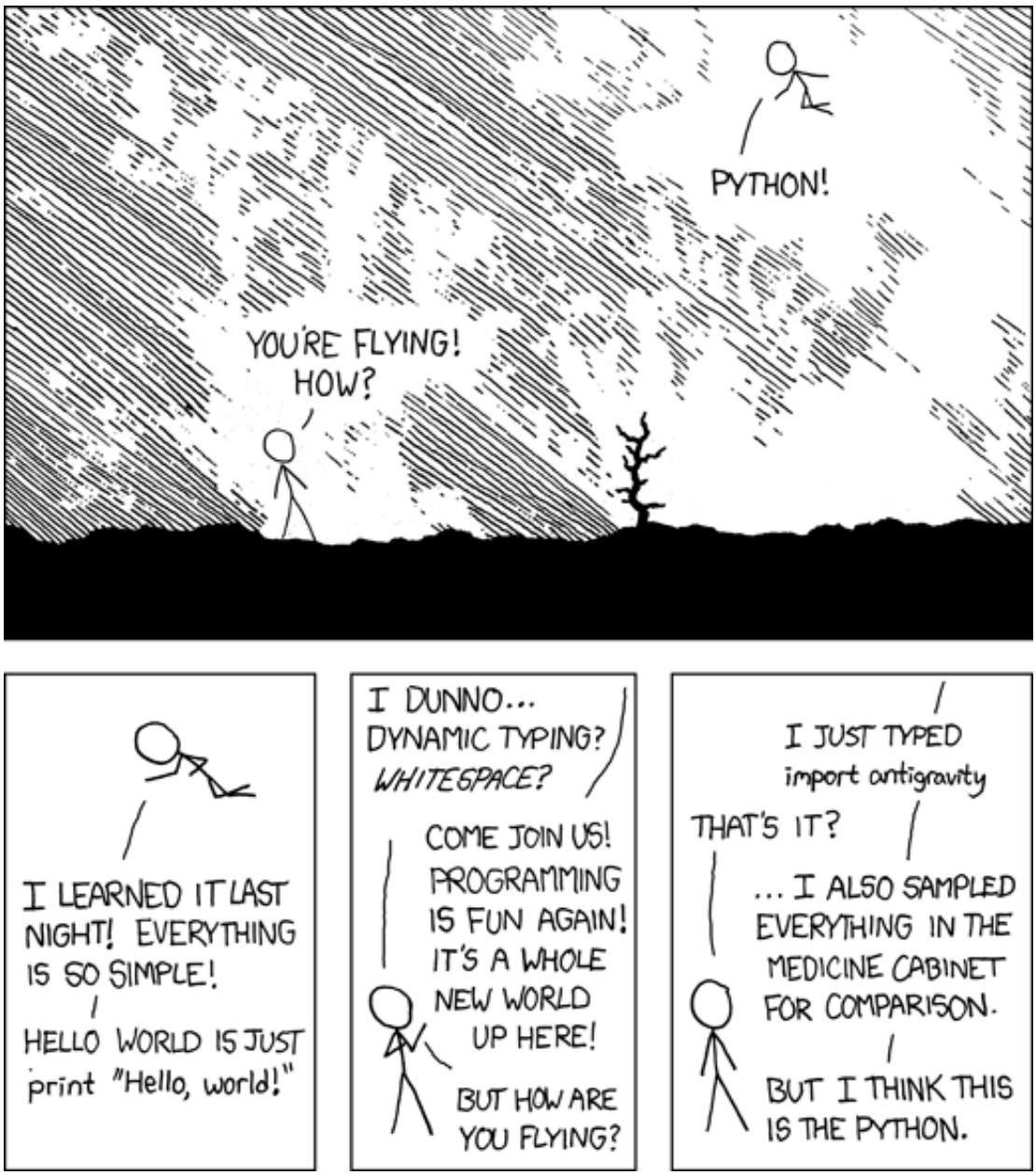
```
import this
```

```
The Zen of Python, by Tim Peters


Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.
```

```
Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do
it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!
```

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

```
import antigravity
```



Модуль math

Давайте вернемся к встроенному модулю `math` и посмотрим внимательнее на то, какие возможности он нам предоставляет.

Мы писали программу для вычисления факториала числа, оказывается, такая функция уже есть:

```
import math



print(math.factorial(5))
```

```
120
```

Довольно часто применяется и функция `gcd` для нахождения наибольшего общего делителя:

```
import math



print(math.gcd(500, 600))
```

```
100
```

Кроме того, есть функции для возведения в степень pow, принимающая число, которое надо возвести в степень, первым аргументом, а вторым аргументом — показатель степени, в которую надо возвести первый аргумент. А для извлечения квадратного корня числа есть функция `math.sqrt`.

В Python есть встроенная функция pow или **, которая также возводит в степень, но есть ряд отличий. Например, math.pow всегда возвращает результат типа float.

```
import math



# Возведение в степень

print(math.pow(10, 10))

# Извлечение квадратного корня

print(math.sqrt(16))
```

```
10000000000.0

4.0
```

В модуль `math` встроены тригонометрические функции вычисления синуса, косинуса, тангенса и т. д.:

```
import math



print(math.sin(math.radians(90)))

print(math.cos(math.radians(0)))

print(math.tan(math.radians(45)))
```

```
1.0

1.0

0.9999999999999999
```

Обратите внимание: они принимают на вход значение угла в радианах, поэтому данные в градусах надо перевести в радианы с помощью

функции модуля `math`, которая называется `radians`. Существует и зеркальная функция degrees для перевода радиан в градусы.

```
import math



print(math.degrees(math.pi))
```

```
180.0
```

Кроме того, `math` содержит ряд дополнительных интересных функций, например, знает теорему Пифагора. Функция `hypot(a, b)` возвращает длину гипотенузы по двум катетам прямоугольного треугольника. Полный перечень функций и их описания можно посмотреть в документации.

Округление чисел

В Python встроены две функции, позволяющие округлять числа: int, отбрасывающая дробную часть числа, и round, использующая банковское округление с указанной точностью. В math есть еще три функции для округления вещественных чисел.

math.ceil

Округление вверх — возвращает ближайшее целое число, большее или равное значению аргумента.

```
from math import ceil



print(ceil(3.14))  # 4
```

math.floor

Округление вниз — возвращает ближайшее целое число, меньшее или равное значению аргумента.

```
from math import floor



print(floor(3.75))  # 3
```

math.trunc

Отбрасывает дробную часть вещественного числа и возвращает его целую часть:

```
from math import trunc



print(trunc(10.834))  # 10
```

Логарифмы

Логарифм числа по определенному основанию позволяет определить степень, в которую нужно возвести основание, чтобы получить это число.

Например, если $2^{10} = 1024$, то логарифм числа 1024 по основанию 2 будет равен 10. Записывается это так:

$$\log_2 1024 = 10$$

Для вычисления логарифмов по основаниям 2 и 10 в библиотеке math есть специальные функции:

```
import math



print(math.log2(65536))  # 16.0

print(math.log10(10 ** 23))  # 23.0
```

Модуль random

Этот модуль предназначен для работы с псевдослучайными последовательностями. Такие последовательности важны в математическом моделировании, в криптографии и 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',
'_ceil',

 '_cos', '_e', '_exp','_inst', '_log', '_pi', '_random', '_sha512',
'_sin',

 '_sqrt', '_test','_test_generator', '_urandom', '_warn',
'betavariate',

 'choice', 'expovariate','gammavariate', 'gauss', 'getrandbits',
'getstate',
```

```
 'lognormvariate','normalvariate', 'paretovariate', 'randint',
'random',

 'randrange', 'sample','seed', 'setstate', 'shuffle', 'triangular',
'uniform',

 'vonmisesvariate','weibullvariate']
```

Как видим, довольно много функций. Давайте рассмотрим некоторые из них.

Для получения одного псевдослучайного целого числа можно воспользоваться одной из двух функций: `randrange` или `randint`. Функция `randrange` возвращает случайное число из диапазона. Как и в обычном `range`, мы можем указать начало, конец и шаг диапазона. Функция `randint` работает похожим образом, но у нее границы диапазона — обязательные параметры, нельзя указать шаг, и верхняя граница включена в диапазон генерации.

```
from random import randrange, randint


# возвращаем случайное целое из диапазона

print(randrange(100))

print(randrange(40, 100, 5))

print(randint(10, 20))
```

Функция choice

Одна из самых популярных — функция `choice`. С её помощью можно выбрать один вариант из нескольких альтернатив, заданных в списке, кортеже, строке или любом другом индексируемом типе. `choice` нельзя применять для неупорядоченных коллекций — например, множеств и словарей.

Например, вот так можно моделировать подкидывание монетки:

```
from random import choice


print(choice((1, 2)))

print(choice(["орел", "решка"]))

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

```
2

'орел'

'a'
```

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))
```

```
2 5

6 5

6 1

1 2

5 6

6 1

4 2

4 2

2 3
```

Если задать символы на сторонах кубика с использованием кодировки Unicode, все будет еще 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.

```
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)
```

Обратите внимание: если попробовать вызвать `choice` с пустой коллекцией, ваша программа упадет с ошибкой. Поэтому перед использованием этой функции будет нелишним проверять наличие в коллекции хотя бы одного элемента.

```
from random import choice

my_str = "Hello, world"
if my_str:
    print(choice(my_str), choice(my_str))
```

Если нам нужно вернуть не один, а несколько элементов, на помощь придут функции `choices` и `sample`. `choices` возвращает заданное именованным параметром k количество элементов с возможными повторами (коллекция должна быть непустой), `sample` — без повторов, но выборка должна быть меньше или равна длине коллекции, иначе тоже будет ошибка.

```
from random import choices, sample

my_list = ['Yes', 'No', 'May be']
```

```
# выбираем k элементов коллекции с повторениями

print(choices(my_list, k=5))

# выбираем k элементов без повторений

print(sample(range(10), 6))
```

```
['No', 'Yes', 'Yes', 'Yes', 'No']

[8, 9, 1, 6, 7, 5]
```

Функция `shuffle` перемешивает список, при этом меняется сам список, который передается как аргумент функции.

```
from random import shuffle



a = list(range(100))

shuffle(a) # меняет сам список

print(a[:10])
```

```
[12, 97, 67, 13, 68, 58, 87, 92, 32, 40]
```

А функция `random` возвращает случайное вещественное число от 0 до 1 (не включительно):

```
from random import random as rnd



print(rnd(), rnd(), rnd()) # вещественное число [0, 1)
```

```
0.7807663953103449 0.1503300563891775 0.6068329639725171
```

В таблице приведен небольшой справочник по наиболее часто используемым функциям из модуля random:

randrange(start, stop[, step])	Возвращает случайное целое число из диапазона start <= N < stop с указанным шагом	`from ran`
		`print(*[`
		`in range`
		`14 2 11 8`

randint(a, b)	Возвращает случайное целое число из диапазона a <= N <= b	`from ran`

		print(*[range(4) 4 5 15 2
random()	Возвращает случайное вещественное число из диапазона [0.0, 1.0)	from ran print(*[in range 0.266 0.4
uniform(a, b)	Возвращает случайное вещественное число из диапазона [a, b], если a <= b, и [b, a], если b < a	from ran print(*[3) for _ 6.357 9.1
choice(seq)	Возвращает случайный элемент из непустой последовательности	from ran print(*[in range c e h d
choices(population, weights=None, *, cum_weights=None, k=1)	Возвращает список длины k с перестановками из непустой последовательности; можно указать относительный вес элемента для выбора	from ran print(*c h e h d
shuffle(seq)	Перемешивает последовательность случайным образом	from ran line = [shuffle(print(li ['d', 'b'

sample(population, k)	Возвращает список из k уникальных элементов последовательности или множества	`from ran`
		
		`print(*s`
		`d c b e`