* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/123/groups/568)

[Урок Библиотеки №1](https://lyceum.yandex.ru/courses/123/groups/568/lessons/641)

**Библиотеки Python. Часть № 1 (random)**

**План урока**

1

[Библиотеки как наследие](https://lyceum.yandex.ru/courses/123/groups/568/lessons/641/materials/1449#1)

2

[Репозиторий PyPI](https://lyceum.yandex.ru/courses/123/groups/568/lessons/641/materials/1449#2)

3

[Встроенные модули](https://lyceum.yandex.ru/courses/123/groups/568/lessons/641/materials/1449#3)

4

[Модуль random](https://lyceum.yandex.ru/courses/123/groups/568/lessons/641/materials/1449#4)

**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 начнём знакомство со стандартными библиотеками языка Python.*

**1. 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.

**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 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 ребёнком, 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.

**2. Репозиторий 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:

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, что количество модулей уже превысило **150 000**!

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 разберёмся со встроенными модулями.

**3. 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**

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

>>> 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, 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

**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(...)

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

>>> 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']

>>> 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))

0.49999999999999994

Обратите внимание, что часть имен начинается с символа «\_». Это служебные имена — мы их пока рассматривать не будем, да и программисты ими пользуются редко.

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.

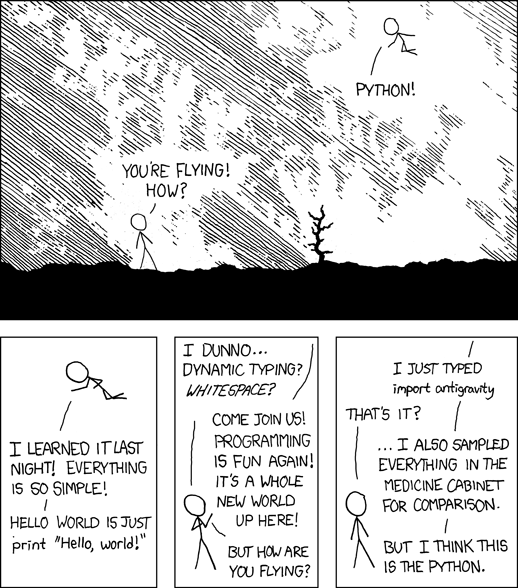
If the implementation is hard to explain, it's a bad idea.

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



**4. 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

>>> 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.

**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 итерируемом типе.

Например, вот так можно моделировать подкидывание монетки:

>>> **from** random **import** choice

>>> choice((1, 2))

2

>>> choice(["орел", "решка"])

'орел'

>>> choice("ab")

'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 Юникод, всё будет ещё реалистичнее.

>>> **from** random **import** choice

>>>

>>> dashes = ['**\u**2680', '**\u**2681', '**\u**2682', '**\u**2683', '**\u**2684', '**\u**2685']

>>> **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)

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты