Лекция 6.3. Стандартные функции. Строковые функции. Лямбда-функции

# **Стандартные функции. Функция main.**

В Python существуют десятки встроенных функций и классов, сотни инструментов, входящих в стандартную библиотеку Python, и тысячи сторонних библиотек на PyPI. Держать всё в голове начинающему программисту нереально. В статье расскажем про стандартные встроенные функции Python: какие используются часто, а какие вам, вероятно, не пригодятся никогда.

Чтобы разобраться, на какие функции стоит обратить внимание, их следует разделить на группы:

* общеизвестные: почти все новички используют эти функции и довольно часто;
* неочевидные для новичков: об этих функциях полезно знать, но их легко пропустить, когда вы новичок в Python;
* понадобятся позже: об этих встроенных функциях полезно помнить, чтобы найти их потом, когда/если они понадобятся;
* можно изучить когда-нибудь: это может пригодиться, но только при определённых обстоятельствах;
* скорее всего, они вам не нужны: они вряд ли понадобятся, если вы не занимаетесь чем-то достаточно специализированным.

Встроенные функции в первых двух категориях являются основными. Они в конечном итоге будут нужны почти всем начинающим программистам на Python. Встроенные модули в следующих двух категориях являются специализированными, но потребности в них будут варьироваться в зависимости от вашей специализации. Категория 5 — это скрытые встроенные функции. Они очень полезны, когда в них есть необходимость, но многим программистам Python они, вероятно, никогда не понадобятся.

## Общеизвестные функции

Если вы уже писали код на Python, эти модули должны быть вам знакомы.

### print

Вряд ли найдётся разработчик, даже начинающий, который не знает эту функцию вывода. Как минимум реализация [«Hello, World!»](https://ru.wikipedia.org/wiki/Hello,_world!) требует использования данного метода. Но вы можете не знать о различных аргументах, которые можно передавать в print.

>>> words = ["Welcome", "to", "Python"]

>>> print(words)

['Welcome', 'to', 'Python']

>>> print(\*words, end="!\n")

Welcome to Python!

>>> print(\*words, sep="\n")

Welcome

to

Python

### len

В Python нет синтаксиса вроде my\_list.length() или my\_string.length, вместо этого используются поначалу непривычные конструкции len(my\_list) и len(my\_string).

>>> words = ["Welcome", "to", "Python"]

>>> len(words)

3

Нравится вам такая реализация или нет, другой альтернативы не предусмотрено, поэтому к ней нужно привыкнуть.

### str

К сожалению, в отличие от многих других языков программирования, в Python нельзя объединять строки и числа.

>>> version = 3

>>> "Python " + version

Traceback (most recent call last):

File "", line 1, in

TypeError: can only concatenate str (not "int") to str

Python отказывается приводить целое число 3 к типу строка, поэтому нужно сделать это самостоятельно, используя встроенную функцию str (технически это класс, но с целью уменьшить количество ненужной информации будем принимать все методы за функции).

>>> version = 3

>>> "Python " + str(version)

'Python 3'

### int

Если нужно пользовательский ввод преобразовать в integer, эта функция незаменима. Она может преобразовывать строки в целые числа.

>>> program\_name = "Python 3"

>>> version\_number = program\_name.split()[-1]

>>> int(version\_number)

3

Эту функцию также можно использовать для отсечения дробной части у числа с плавающей точкой.

>>> from math import sqrt

>>> sqrt(28)

5.291502622129181

>>> int(sqrt(28))

5

Обратите внимание, если нужно обрезать дробную часть при делении, оператор «//» более уместен (с отрицательными числами это работает иначе).

int (3/2) == 3 // 2

### float

Если строка, которую надо конвертировать в число, не является целым числом, здесь поможет метод float.

>>> program\_name = "Python 3"

>>> version\_number = program\_name.split()[-1]

>>> float(version\_number)

3.0

>>> pi\_digits = '3.141592653589793238462643383279502884197169399375'

>>> len(pi\_digits)

50

>>> float(pi\_digits)

3.141592653589793

Float также можно использовать для преобразования целых чисел в числа с плавающей запятой.

В Python 2 такое преобразование необходимо, но в Python 3 целочисленное деление больше не является чем-то особенным (если вы специально не используете оператор «//»). Поэтому больше не нужно использовать float для этой цели, теперь float(x)/y можно легко заменить на x/y.

### list

Эта функция может очень облегчить задачу, если вы хотите составить список из итераций цикла.

>>> numbers = [2, 1, 3, 5, 8]

>>> squares = (n\*\*2 for n in numbers)

>>> squares

at 0x7fd52dbd5930>

>>> list\_of\_squares = list(squares)

>>> list\_of\_squares

[4, 1, 9, 25, 64]

При работе со списком метод copy позволяет создать его копию.

>>> copy\_of\_squares = list\_of\_squares.copy()

Если вы не знаете, с какими элементами работаете, функция list является более общим способом перебора элементов и их копирования.

>>> copy\_of\_squares = list(list\_of\_squares)

Также можно использовать списковое включение, но делать это не рекомендуется.

Обратите внимание, когда вы хотите создать пустой список, следует использовать буквальный синтаксис списка («[ ]»).

>>> my\_list = list() # Так делать нельзя

>>> my\_list = [] # Так можно

Использование «[ ]» считается более идиоматическим, так как эти скобки на самом деле выглядят как список Python.

### tuple

Эта функция во многом похожа на функцию list, за исключением того, что вместо списков она создает кортежи.

>>> numbers = [2, 1, 3, 4, 7]

>>> tuple(numbers)

(2, 1, 3, 4, 7)

Если вы пытаетесь создать хешируемую коллекцию (например, ключ словаря), стоит отдать предпочтению кортежу вместо списка.

### dict

Эта функция создаёт новый словарь.

Подобно спискам и кортежам, dict эквивалентна проходу по массиву пар «ключ-значение» и созданию из них словаря.

Дан список кортежей, по два элемента в каждом.

>>> color\_counts = [('red', 2), ('green', 1), ('blue', 3), ('purple', 5)]

Выведем его на экран с помощью цикла.

>>> colors = {}

>>> for color, n in color\_counts:

... colors[color] = n

...

>>> colors

{'red': 2, 'green': 1, 'blue' 3, 'purple': 5}

То же самое, но с использованием dict.

>>> colors = dict(color\_counts)

>>> colors

{'red': 2, 'green': 1, 'blue' 3, 'purple': 5}

Функция dict может принимать 2 типа аргументов:

* другой словарь: в этом случае этот словарь будет скопирован;
* список кортежей «ключ-значение»: в этом случае из этих слов будет создан новый словарь.

Поэтому следующий код также будет работать.

>>> colors

{'red': 2, 'green': 1, 'blue' 3, 'purple': 5}

>>> new\_dictionary = dict(colors)

>>> new\_dictionary

{'red': 2, 'green': 1, 'blue' 3, 'purple': 5}

Функция dict также может принимать ключевые слова в качестве аргументов для создания словаря со строковыми ключами.

>>> person = dict(name='Trey Hunner', profession='Python Trainer')

>>> person

{'name': 'Trey Hunner', 'profession': 'Python Trainer'}

Но рекомендуется всё же использовать литералы вместо ключевых слов.

>>> person = {'name': 'Trey Hunner', 'profession': 'Python Trainer'}

>>> person

{'name': 'Trey Hunner', 'profession': 'Python Trainer'}

Такой синтаксис более гибок и немного быстрее. Но самое главное он более чётко передаёт факт того, что вы создаёте именно словарь.

Как в случае со списком и кортежем, пустой словарь следует создавать с использованием буквального синтаксиса («{ }»).

>>> my\_list = dict() # Так делать нельзя

>>> my\_list = {} # Так можно

Использование «{ }» более идиоматично и эффективно с точки зрения использования процессора. Обычно для создания словарей используются фигурные скобки, dict встречается гораздо реже.

### set

Функция set создаёт новый набор. Она принимает итерации из хешируемых значений (строк, чисел или других неизменяемых типов) и возвращает set.

>>> numbers = [1, 1, 2, 3, 5, 8]

>>> set(numbers)

{1, 2, 3, 5, 8}

Создать пустой набор с «{ }» нельзя (фигурные скобки создают пустой словарь). Поэтому функция set — лучший способ создать пустой набор.

>>> numbers = set()

>>> numbers

set()

Можно использовать и другой синтаксис.

>>> {\*()} # Так можно создать пустой набор

set()

Такой способ имеет недостаток — он сбивает с толку (он основан на редко используемой функции оператора \*), поэтому он не рекомендуется.

[Звёздный Python: где и как используются \* и \*\*](https://tproger.ru/translations/asterisks-in-python-what-they-are-and-how-to-use-them" \t "_blank)

[tproger.ru](https://tproger.ru/translations/asterisks-in-python-what-they-are-and-how-to-use-them" \t "_blank)

### range

Эта функция создаёт объект range, который представляет собой диапазон чисел.

>>> range(10\_000)

range(0, 10000)

>>> range(-1\_000\_000\_000, 1\_000\_000\_000)

range(-1000000000, 1000000000)

Результирующий диапазон чисел включает начальный номер, но исключает конечный (range(0, 10) не включает 10).

Данная функция полезна при переборе чисел.

>>> for n in range(0, 50, 10):

... print(n)

...

0

10

20

30

40

Обычный вариант использования — выполнить операцию n раз.

first\_five = [get\_things() for \_ in range(5)]

Функция range в Python 2 возвращает список. Это означает, что примеры кода выше будут создавать очень большие списки. Range в Python 3 работает как xrange в Python 2. Числа вычисляются «более лениво» при проходе по диапазону.

## Функции, неочевидные для новичков

### bool

Эта функция проверяет достоверность (истинность) объектов Python. Относительно чисел будет выполняться проверка на неравенство нулю.

>>> bool(5)

True

>>> bool(-1)

True

>>> bool(0)

False

Применяя bool к коллекциям, будет проверяться их длина (больше 0 или нет).

>>> bool('hello')

True

>>> bool('')

False

>>> bool(['a'])

True

>>> bool([])

False

>>> bool({})

False

>>> bool({1: 1, 2: 4, 3: 9})

True

>>> bool(range(5))

True

>>> bool(range(0))

False

>>> bool(None)

False

Проверка истинности очень важна в Python. Вместо того, чтобы задавать вопросы о длине контейнера, многие новички задают проверку истинности.

# Вместо этого

if len(numbers) == 0:

print("The numbers list is empty")

# многие делают так

if not numbers:

print("The numbers list is empty")

Данная функция используется редко. Но, если нужно привести значение к логическому типу для проверки его истинности, bool вам необходима.

### enumerate

Если нужно в цикле посчитать количество элементов (по одному элементу за раз), эта функция может быть очень полезной. Такая задача может показаться специфической, но она бывает нужна довольно часто.

Например, если нужно отслеживать номер строки в файле.

>>> with open('hello.txt', mode='rt') as my\_file:

... for n, line in enumerate(my\_file, start=1):

... print(f"{n:03}", line)

...

001 This is the first line of the file

002 This is the second line

003 This is the last line of the file

Enumerate также часто используется для отслеживания индекса элементов в последовательности.

def palindromic(sequence):

"""Возвращает True, если последовательность является палиндромом."""

for i, item in enumerate(sequence):

if item != sequence[-(i+1)]:

return False

return True

Также следует обратить внимание, что новички в Python часто используют range(len(sequence)). Если вы когда-нибудь встретите конструкции типа range(len(...)), лучше заменить её на enumerate. Она поможет упростить конструкцию операторов.

def palindromic(sequence):

"""Возвращает True, если последовательность является палиндромом."""

for i in range(len(sequence)):

if sequence[i] != sequence[-(i+1)]:

return False

return True

### zip

Эта функция ещё более специализирована, чем enumerate. Zip используется для перебора сразу нескольких объектов одновременно.

>>> one\_iterable = [2, 1, 3, 4, 7, 11]

>>> another\_iterable = ['P', 'y', 't', 'h', 'o', 'n']

>>> for n, letter in zip(one\_iterable, another\_iterable):

... print(letter, n)

...

P 2

y 1

t 3

h 4

o 7

n 11

По сравнению с enumerate, последняя функция удобна, когда нужна индексация во время цикла. Если нужно обрабатывать несколько объектов одновременно, zip предпочтительнее enumerate.

### reversed

Функция reversed, как enumerate и zip, возвращает итератор.

>>> numbers = [2, 1, 3, 4, 7]

>>> reversed(numbers)

<list\_reverseiterator object at 0x7f3d4452f8d0>

Единственное, что можно сделать с этим итератором, пройтись по нему (но только один раз).

>>> reversed\_numbers = reversed(numbers)

>>> list(reversed\_numbers)

[7, 4, 3, 1, 2]

>>> list(reversed\_numbers)

[]

Подобно enumerate и zip, reversed является своего рода вспомогательной функцией в циклах. Её использование можно увидеть исключительно в цикле for.

>>> for n in reversed(numbers):

... print(n)

...

7

4

3

1

2

Есть несколько и других способов перевернуть списки в Python.

# Синтаксис нарезки

for n in numbers[::-1]:

print(n)

# Метод переворота на месте

numbers.reverse()

for n in numbers:

print(n)

Данная функция, как правило, является лучшим способом «перевернуть» любой список (а также набор, массив и т. д.) в Python.

В отличие от numbers.reverse(), reversed не изменяет список, а возвращает итератор перевёрнутых элементов.

reversed(numbers) (в отличие от numbers [:: - 1]) не создает новый список. Возвращаемый им итератор извлекает следующий элемент в обратном порядке при проходе по циклу. Также синтаксис reversed(numbers) намного более читабелен, чем numbers [:: - 1].

Можно использовать факт, что reversed не копирует список. Если объединить его с функцией zip, можно переписать функцию palindromic (из раздела enumerate), не занимая дополнительной памяти (не копируя объект).

def palindromic(sequence):

"""Возвращает True, если последовательность является палиндромом."""

for n, m in zip(sequence, reversed(sequence)):

if n != m:

return False

return True

### sum

Эта функция берёт набор чисел и возвращает их сумму.

>>> sum([2, 1, 3, 4, 7])

17

В Python есть много вспомогательных функций, которые выполняют циклы за вас (отчасти потому, что они хорошо сочетаются с генератор-выражениями).

>>> numbers = [2, 1, 3, 4, 7, 11, 18]

>>> sum(n\*\*2 for n in numbers)

524

### min и max

Эти функции выдают минимальное и максимальное число из набора соответственно.

>>> numbers = [2, 1, 3, 4, 7, 11, 18]

>>> min(numbers)

1

>>> max(numbers)

18

Данные методы сравнивают элементы, используя оператор <. Поэтому, все передаваемые в них значения должны быть упорядочены и сопоставимы друг с другом.

Min и max также принимают key-свойство, позволяющее настроить, что на самом деле означают «минимум» и «максимум» для конкретных объектов.

>>> fruits = ['kumquat', 'Cherimoya', 'Loquat', 'longan', 'jujube']

>>> sorted(fruits, key=len)

['Loquat', 'longan', 'jujube', 'kumquat', 'Cherimoya']

### sorted

Эта функция принимает любой набор элементов и возвращает новый список всех значений в отсортированном порядке.

>>> numbers = [1, 8, 2, 13, 5, 3, 1]

>>> words = ["python", "is", "lovely"]

>>> sorted(words)

['is', 'lovely', 'python']

>>> sorted(numbers, reverse=True)

[13, 8, 5, 3, 2, 1, 1]

Данная функция (как min и max) сравнивает элементы, используя оператор <, поэтому все значения, переданные ей, должны быть упорядочены.

Sorted также позволяет настраивать сортировку с помощью key-свойства.

### any и all

Эти функции могут быть использованы в паре с генератор-выражениями, чтобы определить соответствие элементов заданному условию.

Используя all, можно переписать функцию palindromic следующим образом.

def palindromic(sequence):

"""Возвращает True, если последовательность является палиндромом."""

return all(

n == m

for n, m in zip(sequence, reversed(sequence))

)

Отрицание условия и возвращаемого значения позволит также использовать any в этом примере точно также (что усложнит конструкцию, но вполне сойдёт в качестве примера использования).

def palindromic(sequence):

"""Возвращает True, если последовательность является палиндромом."""

return not any(

n != m

for n, m in zip(sequence, reversed(sequence))

)

## 5 функций для отладки

Эти функции часто игнорируются, но будут полезны для отладки и устранения неисправностей кода.

### breakpoint

Если нужно приостановить выполнение кода и перейти в командную строку Python, эта функция вам пригодится. Вызов breakpoint перебросит вас в отладчик Python.

Эта встроенная функция была добавлена в Python 3.7, но если вы работаете в более старых версиях, можете получить тот же результат с помощью import pdb; pdb.set\_trace().

### dir

Эта функция может использоваться в двух случаях:

* просмотр списка всех локальных переменных;
* просмотр списка всех атрибутов конкретного объекта.

Из примера можно увидеть локальные переменные сразу после запуска и после создания новой переменной x.

>>> dir()

['\_\_annotations\_\_', '\_\_doc\_\_', '\_\_name\_\_', '\_\_package\_\_']

>>> x = [1, 2, 3, 4]

>>> dir()

['\_\_annotations\_\_', '\_\_doc\_\_', '\_\_name\_\_', '\_\_package\_\_', 'x']

Если в dir передать созданный список x, на выходе можно увидеть все его атрибуты.

>>> dir(x)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

В выведенном списке атрибутов можно увидеть его типичные методы (append, pop, remove и т. д.) , а также множество более сложных методов для перегрузки операторов.

### vars

Эта функция является своего рода смесью двух похожих инструментов: locals() и \_\_dict\_\_.

Когда vars вызывается без аргументов, это эквивалентно вызову locals(), которая показывает словарь всех локальных переменных и их значений.

>>> vars()

{'\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None, '\_\_package\_\_': None, '\_\_loader\_\_': <class '\_frozen\_importlib.BuiltinImporter'>, '\_\_spec\_\_': None, '\_\_annotations\_\_': {}, '\_\_builtins\_\_': <module 'builtins' (built-in)>}

Когда вызов происходит с аргументом, vars получает доступ к атрибуту \_\_dict\_\_, который представляет собой словарь всех атрибутов экземпляра.

>>> from itertools import chain

>>> vars(chain)

mappingproxy({'\_\_getattribute\_\_': <slot wrapper '\_\_getattribute\_\_' of 'itertools.chain' objects>, '\_\_iter\_\_': <slot wrapper '\_\_iter\_\_' of 'itertools.chain' objects>, '\_\_next\_\_': <slot wrapper '\_\_next\_\_' of 'itertools.chain' objects>, '\_\_new\_\_': <built-in method \_\_new\_\_ of type object at 0x5611ee76fac0>, 'from\_iterable': <method 'from\_iterable' of 'itertools.chain' objects>, '\_\_reduce\_\_': <method '\_\_reduce\_\_' of 'itertools.chain' objects>, '\_\_setstate\_\_': <method '\_\_setstate\_\_' of 'itertools.chain' objects>, '\_\_doc\_\_': 'chain(\*iterables) --> chain object\n\nReturn a chain object whose .\_\_next\_\_() method returns elements from the\nfirst iterable until it is exhausted, then elements from the next\niterable, until all of the iterables are exhausted.'})

Перед использованием vars было бы неплохо сначала обратиться к dir.

### type

Эта функция возвращает тип объекта, который вы ей передаете.

Тип экземпляра класса есть сам класс.

>>> x = [1, 2, 3]

>>> type(x)

<class 'list'>

Тип класса — это его метакласс, обычно это type.

>>> type(list)

<class 'type'>

>>> type(type(x))

<class 'type'>

Атрибут \_\_class\_\_ даёт тот же результат, что и функция type, но рекомендуется использовать второй вариант.

>>> x.\_\_class\_\_

<class 'list'>

>>> type(x)

<class 'list'>

Функция type, кроме отладки, иногда полезна и в реальном коде (особенно в объектно-ориентированном программировании с наследованием и пользовательскими строковыми представлениями).

Обратите внимание, что при проверке типов обычно вместо type используется функция isinstance. Также стоит понимать, что в Python обычно не принято проверять типы объектов (вместо этого практикуется [утиная типизация](https://ru.wikipedia.org/wiki/Утиная_типизация)).

### help

Если вы находитесь в Python Shell или делаете отладку кода с использованием breakpoint, и хотите знать, как работает определённый объект, метод или атрибут, функция help поможет вам.

В действительности вы, скорее всего, будете обращаться за помощью к поисковой системе. Но если вы уже находитесь в Python Shell, вызов help(list.insert) будет быстрее, чем поиск документации в Google.

## Функции, которые пригодятся позже

В начале изучения Python эти функции вам по большей части будут не нужны, но в конечном итоге они вам понадобятся.

### open

Эта функция служит для открытия файла и последующей работы с ним. Но, если вы не работаете с файлами напрямую, то она вам может и не пригодиться.

Несмотря на то, что работа с файлами очень распространена, далеко не все программисты Python работают с файлами через open. Например, разработчики Django могут не использовать её вообще.

### input

Эта функция запрашивает у пользователя ввод, ждёт нажатия клавиши Enter, а затем возвращает набранный текст.

Чтение из стандартного ввода — это один из способов получить входные данные в программе. Но есть и много других способов: аргументы командной строки, чтение из файла, чтение из базы данных и многое другое.

### repr

Эта функция необходима для представления объекта в читабельном виде.

Для многих объектов функции str и repr работают одинаково.

>>> str(4), repr(4)

('4', '4')

>>> str([]), repr([])

('[]', '[]')

Но есть объекты, для которых их применение различается.

>>> str('hello'), repr("hello")

('hello', "'hello'")

>>> from datetime import date

>>> str(date(2020, 1, 1)), repr(date(2020, 1, 1))

('2020-01-01', 'datetime.date(2020, 1, 1)')

Строковое представление, которое вы видите в Python Shell, использует repr, тогда как функция print использует str.

>>> date(2020, 1, 1)

datetime.date(2020, 1, 1)

>>> "hello!"

'hello!'

>>> print(date(2020, 1, 1))

2020-01-01

>>> print("hello!")

hello!

Также repr используется при ведении лог-журнала, обработке исключений и реализации более сложных методов.

### super

Эта функция очень важна, если используется наследование одного класса от другого.

Многие пользователи Python редко создают классы. Они не являются важной частью Python, хоть для многих типов программирования они необходимы. Например, вы не можете использовать веб-фреймворк Django без создания классов.

### property

Эта функция является [декоратором](https://docs.python.org/3/glossary.html#term-decorator) и [дескриптором](https://docs.python.org/3/glossary.html#term-descriptor).

[Декораторы в Python: понять и полюбить](https://tproger.ru/translations/demystifying-decorators-in-python" \t "_blank)

[tproger.ru](https://tproger.ru/translations/demystifying-decorators-in-python" \t "_blank)

Декоратор позволяет создать атрибут, который всегда будет содержать возвращаемое значение конкретного вызова функции. Это проще всего понять на примере.

Пример класса, который использует property.

class Circle:

def \_\_init\_\_(self, radius=1):

self.radius = radius

@property

def diameter(self):

return self.radius \* 2

Здесь вы можете увидеть доступ к атрибуту diameter объекта Circle.

>>> circle = Circle()

>>> circle.diameter

2

>>> circle.radius = 5

>>> circle.diameter

10

Если вы занимаетесь объектно-ориентированным программированием на Python, вам, вероятно, захочется узнать о property больше в какой-то момент. В отличие от других объектно-ориентированных языков, в Python property используется вместо методов getter и setter.

### issubclass и isinstance

Функция issubclass проверяет, является ли класс подклассом одного или нескольких других классов.

>>> issubclass(int, bool)

False

>>> issubclass(bool, int)

True

>>> issubclass(bool, object)

True

Функция isinstance проверяет, является ли объект экземпляром одного или нескольких классов.

>>> isinstance(True, str)

False

>>> isinstance(True, bool)

True

>>> isinstance(True, int)

True

>>> isinstance(True, object)

True

Функция isinstance может быть представлена как делегирование в issubclass.

>>> issubclass(type(True), str)

False

>>> issubclass(type(True), bool)

True

>>> issubclass(type(True), int)

True

>>> issubclass(type(True), object)

True

Если вы перегружаете операторы, вам может понадобиться использование isinstance, но в целом в Python стараются избегать строгой проверки типов, поэтому особой нужды в данных функциях нет.

### hasattr, getattr, setattr и delattr

Если нужно работать с атрибутами объекта, но имя атрибутов является динамическим и постоянно меняется, данные функции вам будут необходимы.

Например, есть объект thing, который нужно проверить на конкретное значение.

>>> class Thing: pass

...

>>> thing = Thing()

Функция hasattr позволяет проверить, имеет ли объект определённый атрибут.

>>> hasattr(thing, 'x')

False

>>> thing.x = 4

>>> hasattr(thing, 'x')

True

Функция getattr позволяет получить значение атрибута (с необязательным значением по умолчанию, если атрибут не существует).

>>> getattr(thing, 'x')

4

>>> getattr(thing, 'x', 0)

4

>>> getattr(thing, 'y', 0)

0

Функция setattr позволяет установить значение атрибута.

>>> setattr(thing, 'x', 5)

>>> thing.x

5

И delattr соответственно удаляет атрибут.

>>> delattr(thing, 'x')

>>> thing.x

Traceback (most recent call last):

File "", line 1, in

AttributeError: 'Thing' object has no attribute 'x'

### classmethod и staticmethod

Если у вас есть метод, который должен вызываться в экземпляре или в классе, вам нужен декоратор classmethod. Фабричные методы (альтернативные конструкторы) являются распространённым случаем для этого.

class RomanNumeral:

"""Римская цифра, представленная как строка и число."""

def \_\_init\_\_(self, number):

self.value = number

@classmethod

def from\_string(cls, string):

return cls(roman\_to\_int(string)) # Функция пока еще не существует

Немного сложнее придумать хорошее использование staticmethod, так как всегда можно использовать функцию уровня модуля вместо статического метода.

class RomanNumeral:

"""Римская цифра, представленная как строка и число."""

SYMBOLS = {'M': 1000, 'D': 500, 'C': 100, 'L': 50, 'X': 10, 'V': 5, 'I': 1}

def \_\_init\_\_(self, number):

self.value = number

@classmethod

def from\_string(cls, string):

return cls(cls.roman\_to\_int(string))

@staticmethod

def roman\_to\_int(numeral):

total = 0

for symbol, next\_symbol in zip\_longest(numeral, numeral[1:]):

value = RomanNumeral.SYMBOLS[symbol]

next\_value = RomanNumeral.SYMBOLS.get(next\_symbol, 0)

if value < next\_value:

value = -value

total += value

return total

Функция roman\_to\_int не требует доступа к экземпляру или классу, поэтому ей не нужно быть @classmethod. Фактически нет необходимости делать эту функцию staticmethod (вместо classmethod). staticmethod является просто более сдерживающим, чтобы сигнализировать о том факте, что функция не зависима от класса, в котором она находится.

### next

Данная функция возвращает следующий элемент в итераторе.

Она может работать со следующими видами итераторов:

* объекты enumerate;
* объекты zip;
* возвращаемые значения функции reversed;
* файлы
* объекты csv.reader;
* генератор-выражения;
* генератор-функции;

Функция next может быть представлена как способ вручную перебрать набор, чтобы получить один единственный элемент, а затем выйти из перебора.

## Функции, которые когда-нибудь можно выучить

Следующие встроенные функции Python определённо не бесполезны, но они более специализированы.

Эти функции вам, возможно, будут нужны, но также есть шанс, что вы никогда не прибегнете к ним в своём коде.

* [iter](https://docs.python.org/3/library/functions.html#iter): возвращает итератор (список, набор и т. д.);
* [callable](https://docs.python.org/3/library/functions.html#callable): возвращает True, если аргумент является вызываемым;
* [filter](https://docs.python.org/3/library/functions.html#filter) and [map](https://docs.python.org/3/library/functions.html#map): вместо них рекомендуется использовать генератор-выражения;
* [Round](https://docs.python.org/3/library/functions.html#round): округляет число;
* [divmod](https://docs.python.org/3/library/functions.html#divmod): эта функция выполняет деление без остатка (//) и операцию по модулю (%) одновременно;
* [bin](https://docs.python.org/3/library/functions.html#bin), [oct](https://docs.python.org/3/library/functions.html#oct) и [hex](https://docs.python.org/3/library/functions.html#hex): служат для отображения чисел в виде строки в двоичной, восьмеричной или шестнадцатеричной форме;
* [abs](https://docs.python.org/3/library/functions.html#abs): возвращает абсолютное значение числа (аргумент может быть целым или числом с плавающей запятой, если аргумент является комплексным числом, его величина возвращается);
* [hash](https://docs.python.org/3/library/functions.html#hash);
* [object](https://docs.python.org/3/library/functions.html#object).

## Прочие специфические функции

* [ord](https://docs.python.org/3/library/functions.html#ord) и [chr](https://docs.python.org/3/library/functions.html#chr): могут пригодиться при изучении ASCII или Unicode;
* [exec](https://docs.python.org/3/library/functions.html#exec) и [eval](https://docs.python.org/3/library/functions.html#eval): для исполнения строки;
* [compile](https://docs.python.org/3/library/functions.html#compile);
* [slice](https://docs.python.org/3/library/functions.html#slice): если вы реализуете \_\_getitem\_\_ для создания пользовательской последовательности, это может вам понадобиться;
* [bytes](https://docs.python.org/3/library/functions.html#bytes), [bytearray](https://docs.python.org/3/library/functions.html#bytearray) и [memoryview](https://docs.python.org/3/library/functions.html#memoryview): если вы часто работаете с байтами;
* [ascii](https://docs.python.org/3/library/functions.html#ascii): похож на repr, но возвращает представление объекта только в ASCII;
* [frozenset](https://docs.python.org/3/library/functions.html#frozenset): как set, но он неизменен (и хешируемый);
* [\_\_import\_\_](https://docs.python.org/3/library/functions.html#__import__): лучше использовать importlib;
* [format](https://docs.python.org/3/library/functions.html#format): вызывает метод \_\_format\_\_, который используется для форматирования строк;
* [pow](https://docs.python.org/3/library/functions.html#pow): оператор возведения в степень (\*\*);
* [complex](https://docs.python.org/3/library/functions.html#complex): если вы не используете комплексные числа (4j + 3), она вам не понадобится.

## Заключение

Если вы только начинаете свой путь в изучении Python, нет необходимости изучать все встроенные функции сейчас. Не торопитесь, сосредоточьтесь на первых двух пунктах (общеизвестные и упускаемые из виду), а ​​после можете перейти и к другим, если/когда они вам понадобятся.

В различных руководствах по Python часто используется функция main(). Но иногда это может быть и лишним.

Лаконичность — важный элемент программирования. И вполне логично задавать вопрос о том, зачем разработчику добавлять лишние строки в программу. Функция main() требует дополнительных строк, но она служит определенной цели.

В этом материале речь пойдет о функции main(), о том, какую пользу она приносит, а также о том, как правильно использовать ее в коде.

## Что такое функция main()?

Main — это специальная функция, которая позволяет добавить больше логики в структуру программ. В Python можно вызвать функцию в нижней части программы, которая и будет запущена. Рассмотрим такой пример:

Копировать

def cookies():

print("Печенье вкусное!")

print("Дайте печенье.")

cookies()

Запустим этот код:

Дайте печенье.

Печенье вкусное!

## Как создать функцию main

Есть две части основной функции Python. Первая — сама функция main(). В ней хранится код основной программы. Вызовем функцию cookies() и выведем выражение «Дайте печенье.» из функции main():

Копировать

def cookies():

print("Печенье вкусное!")

def main():

print("Дайте печенье.")

cookies()

Теперь вызовем функцию в конце программы:

Копировать

main()

Можно запустить код:

Дайте печенье.

Печенье вкусное!

Результат не поменялся. Но теперь сам код читать легче. Пусть в нем и больше строк (и одна дополнительная строка), вполне очевидно, что именно происходит:

* Cookies() — функция, которая выводит «Печенье вкусное!».
* Когда программа запускается, определяются две функции: cookies() и main().
* Затем функция main() вызывается.
* «Дайте печенье.» выводится в консоли.
* Затем вызывается функция cookies(), которая выводит в консоль «Печенье вкусное!».

Код не только чище, но и логичнее. Если вы пришли из другого языка, например, Java, но знаете, насколько это удобно.

## Значение \_\_name\_\_

Прежде чем переходить к инструкциям \_\_name\_\_ и \_\_main\_\_, которые часто используются вместе, нужно обсудить \_\_name\_\_. \_\_name\_\_ хранит название программы.

Если запустить файл прямо, то значением \_\_name\_\_ будет \_\_main\_\_. Предположим, что файл называется print\_name.py:

Копировать

print(\_\_name\_\_)

Этот код можно запустить следующим образом:

$ python print\_name.py

Он вернет \_\_main\_\_.

Предположим, что этот код был импортирован в качестве модуля в файл main.py:

Копировать

import print\_name

Запустим его:

$ python main.py

Код вернет:

print\_name

Код внутри print\_name.py исполняется потому что он был импортирован в качестве модуля в основной программе. Файл print\_name выводит \_\_name\_\_ в консоль. Поскольку print\_name был импортирован в качестве модуля, то и значением \_\_name\_\_ является print\_name.

## if \_\_name\_\_ == \_\_main\_\_ в Python

Вы наверняка встречали следующую конструкцию в программах на Python в функции main:

Копировать

if \_\_name\_\_ == "\_\_main\_\_":

Что она значит? В Python любая переменная, начинающаяся с двух символов нижнего подчеркивания (\_\_), является специальной. Это зарезервированные значения, которые выполняют определенную роль в программе.

\_\_main\_\_ указывает на область видимости, где будет выполняться код. Если запустить Python-файл прямо, то значением \_\_name\_\_ будет \_\_main\_\_. Если же его запустить в качестве модуля, то значением будет уже не \_\_main\_\_, а название модуля.

Это значит, что строка выше вернет True только в том случае, если программа будет запущена прямо.

Если же ссылаться на файл как на модуль, то содержимое конструкции if не будет выполнено. Рассмотрим на примере.

### Как использовать \_\_name\_\_ и \_\_main\_\_

Создадим новый скрипт на Python под названием username.py. В коде будем просить пользователя ввести его имя и проверять, не является ли его длина больше 5. Если символов не больше 5, то попросим ввести имя снова.

Начнем с определения глобальной переменной для хранения имени пользователя:

Копировать

username = ""

После этого определим две функции. Первая будет просить пользователя ввести имя пользователя и проверять, не больше ли 5 символов в нем. Вторая будет выводить значение в оболочке Python:

Копировать

def choose\_username():

global username

username = input("Введите логин: ")

if len(username) > 5:

print("Ваш логин сохранен.")

else:

print("Пожалуйста, выберите имя пользователя длиной более пяти символов.")

choose\_username()

def print\_username():

print(username)

В этом примере использовалось ключевое слово global, чтобы содержимое, присвоенное в методе choose\_username(), было доступно глобально. После определения функций нужно создать main, которая и будет их вызывать

Копировать

def main():

choose\_username()

print\_username()

После этого нужно добавить if \_\_name\_\_="\_\_main\_\_" в инструкцию if. Это значит, что при запуске файла прямо, интерпретатор Python исполнит две функции. Если же запустить код в качестве модуля, то содержимое внутри main() исполнено не будет.

Копировать

if \_\_name\_\_ == "\_\_main\_\_":

main()

Запустим код:

$ python username.py

Он вернет следующее:

Введите логин: mylogin

Ваш логин сохранен.

mylogin

Этот код запускает функцию choose\_username(), а затем — print\_username(). Если указать имя длиной меньше 4 символов, то ответ будет таким:

Введите логин: Li

Ваш логин сохранен.

Пожалуйста, выберите имя пользователя длиной более пяти символов.

Введите логин:

Будет предложено ввести другое имя. Если импортировать этот код в качестве модуля, то функция main() не запустится.

## Выводы

Функция main() используется для разделения блоков кода в программе. Использование функции main() обязательно в таких языках, как Java, потому что это упрощает понимание того, в каком порядке код запускается в программе. В Python функцию main() писать необязательно, но это улучшает читаемость кода.

# **Строковые функции в Python. Преобразование строк**

Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Это первая часть о работе со строками, а именно о литералах строк.

Литералы строк

Работа со строками в Python очень удобна. Существует несколько литералов строк, которые мы сейчас и рассмотрим.

Строки в апострофах и в кавычках

S = 'spam"s'

S = "spam's"

Строки в апострофах и в кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы позволить вставлять в литералы строк символы кавычек или апострофов, не используя экранирование.

Экранированные последовательности - служебные символы

Экранированные последовательности позволяют вставить символы, которые сложно ввести с клавиатуры.

|  |  |
| --- | --- |
| Экранированная последовательность | Назначение |
| \n | Перевод строки |
| \a | Звонок |
| \b | Забой |
| \f | Перевод страницы |
| \r | Возврат каретки |
| \t | Горизонтальная табуляция |
| \v | Вертикальная табуляция |
| \N{id} | Идентификатор ID базы данных Юникода |
| \uhhhh | 16-битовый символ Юникода в 16-ричном представлении |
| \Uhhhh… | 32-битовый символ Юникода в 32-ричном представлении |
| \xhh | 16-ричное значение символа |
| \ooo | 8-ричное значение символа |
| \0 | Символ Null (не является признаком конца строки) |

"Сырые" строки - подавляют экранирование

Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования отключается.

S = r'C:\newt.txt'

Но, несмотря на назначение, "сырая" строка не может заканчиваться символом обратного слэша. Пути решения:

S = r'\n\n\\'[:-1]

S = r'\n\n' + '\\'

S = '\\n\\n'

Строки в тройных апострофах или кавычках

Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд.

>>> c = '''это очень большая

... строка, многострочный

... блок текста'''

>>> c

'это очень большая\nстрока, многострочный\nблок текста'

>>> print(c)

это очень большая

строка, многострочный

блок текста

## Базовые операции

* Конкатенация (сложение)

>>>

>>> S1 = 'spam'

>>> S2 = 'eggs'

>>> print(S1 + S2)

'spameggs'

* Дублирование строки

>>>

>>> print('spam' \* 3)

spamspamspam

* Длина строки (функция len)

>>>

>>> len('spam')

4

* Доступ по индексу

>>>

>>> S = 'spam'

>>> S[0]

's'

>>> S[2]

'a'

>>> S[-2]

'a'

Как видно из примера, в Python возможен и доступ по отрицательному индексу, при этом отсчет идет от конца строки.

* Извлечение среза

Оператор извлечения среза: [X:Y]. X – начало среза, а Y – окончание;

символ с номером Y в срез не входит. По умолчанию первый индекс равен 0, а второй - длине строки.

>>>

>>> s = 'spameggs'

>>> s[3:5]

'me'

>>> s[2:-2]

'ameg'

>>> s[:6]

'spameg'

>>> s[1:]

'pameggs'

>>> s[:]

'spameggs'

Кроме того, можно задать шаг, с которым нужно извлекать срез.

>>>

>>> s[::-1]

'sggemaps'

>>> s[3:5:-1]

''

>>> s[2::2]

'aeg'

## Другие функции и методы строк

При вызове методов необходимо помнить, что строки в Python относятся к категории неизменяемых последовательностей, то есть все функции и методы могут лишь создавать новую строку.

>>>

>>> s = 'spam'

>>> s[1] = 'b'

Traceback (most recent call last):

File "", line 1, in

s[1] = 'b'

TypeError: 'str' object does not support item assignment

>>> s = s[0] + 'b' + s[2:]

>>> s

'sbam'

Поэтому все строковые методы возвращают новую строку, которую потом следует присвоить переменной.

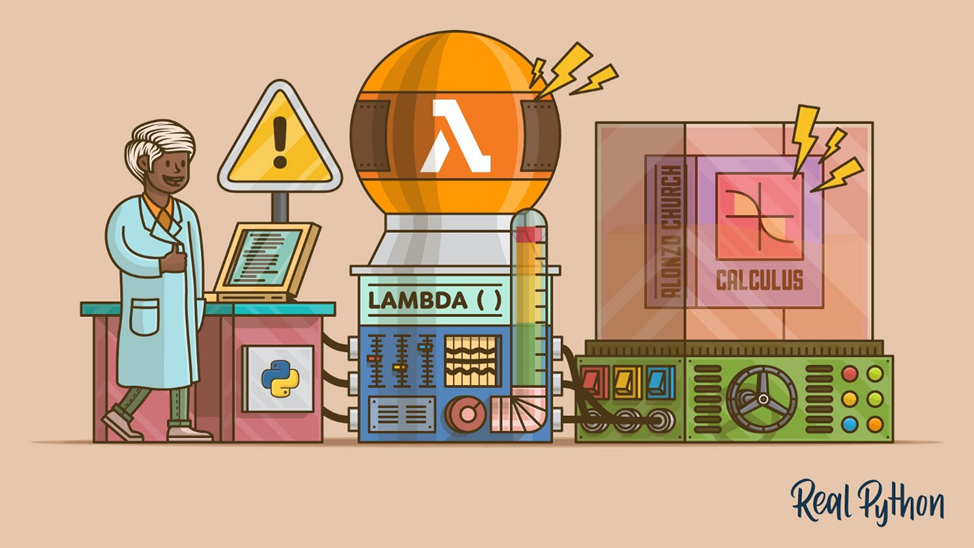
## Таблица "Функции и методы строк"

|  |  |
| --- | --- |
| Функция или метод | Назначение |
| S = 'str'; S = "str"; S = '''str'''; S = """str""" | [Литералы строк](https://pythonworld.ru/tipy-dannyx-v-python/stroki-literaly-strok.html) |
| S = "s\np\ta\nbbb" | Экранированные последовательности |
| S = r"C:\temp\new" | Неформатированные строки (подавляют экранирование) |
| S = b"byte" | Строка [байтов](https://pythonworld.ru/tipy-dannyx-v-python/bajty-bytes-i-bytearray.html) |
| S1 + S2 | Конкатенация (сложение строк) |
| S1 \* 3 | Повторение строки |
| S[i] | Обращение по индексу |
| S[i:j:step] | Извлечение среза |
| len(S) | Длина строки |
| S.find(str, [start],[end]) | Поиск подстроки в строке. Возвращает номер первого вхождения или -1 |
| S.rfind(str, [start],[end]) | Поиск подстроки в строке. Возвращает номер последнего вхождения или -1 |
| S.index(str, [start],[end]) | Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError |
| S.rindex(str, [start],[end]) | Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError |
| S.replace(шаблон, замена[, maxcount]) | Замена шаблона на замену. maxcount ограничивает количество замен |
| S.split(символ) | Разбиение строки по разделителю |
| S.isdigit() | Состоит ли строка из цифр |
| S.isalpha() | Состоит ли строка из букв |
| S.isalnum() | Состоит ли строка из цифр или букв |
| S.islower() | Состоит ли строка из символов в нижнем регистре |
| S.isupper() | Состоит ли строка из символов в верхнем регистре |
| S.isspace() | Состоит ли строка из неотображаемых символов (пробел, символ перевода страницы ('\f'), "новая строка" ('\n'), "перевод каретки" ('\r'), "горизонтальная табуляция" ('\t') и "вертикальная табуляция" ('\v')) |
| S.istitle() | Начинаются ли слова в строке с заглавной буквы |
| S.upper() | Преобразование строки к верхнему регистру |
| S.lower() | Преобразование строки к нижнему регистру |
| S.startswith(str) | Начинается ли строка S с шаблона str |
| S.endswith(str) | Заканчивается ли строка S шаблоном str |
| S.join(список) | Сборка строки из списка с разделителем S |
| ord(символ) | Символ в его код ASCII |
| chr(число) | Код ASCII в символ |
| S.capitalize() | Переводит первый символ строки в верхний регистр, а все остальные в нижний |
| S.center(width, [fill]) | Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию) |
| S.count(str, [start],[end]) | Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию) |
| S.expandtabs([tabsize]) | Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам |
| S.lstrip([chars]) | Удаление пробельных символов в начале строки |
| S.rstrip([chars]) | Удаление пробельных символов в конце строки |
| S.strip([chars]) | Удаление пробельных символов в начале и в конце строки |
| S.partition(шаблон) | Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустых строки |
| S.rpartition(sep) | Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустых строки, а затем саму строку |
| S.swapcase() | Переводит символы нижнего регистра в верхний, а верхнего – в нижний |
| S.title() | Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний |
| S.zfill(width) | Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями |
| S.ljust(width, fillchar=" ") | Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar |
| S.rjust(width, fillchar=" ") | Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar |
| S.format(\*args, \*\*kwargs) | [Форматирование строки](https://pythonworld.ru/osnovy/formatirovanie-strok-metod-format.html) |

# **Лямбда‑функции**

В этой статье вы подробнее изучите анонимные функции, так же называемые "лямбда-функции". Давайте разберемся, что это такое, каков их синтаксис и как их использовать ( с примерами).

Лямбда-функции в Python являются анонимными. Это означает, что функция безымянна. Как известно, ключевое слов def используется в Python для определения обычной функции. В свою очередь, ключевое слово  lambda  используется для определения анонимной функции.



## Лямбда-функция имеет следующий синтаксис.

Lambda аргументы: выражение

Лямбда-функции могут иметь любое количество аргументов, но у каждой может быть только одно выражение. Выражение вычисляется и возвращается. Эти функции могут быть использованы везде, где требуется объект-функция.

### 1.1. Пример лямбда-функции.

Ниже представлен пример лямбда-функции,  удваивающей вводимое значение.

double = lambda x: x\*2  
print(double(5))

**Вывод:**

10

В вышеуказанном коде lambda x: x\*2 — это лямбда-функция. Здесь x— это аргумент,аx\*2 — это выражение, которое вычисляется и возвращается.

Эта функция безымянная. Она возвращает функциональный объект с идентификатором double. Сейчас мы можем считать её обычной функцией.

Инструкция:

double = lambda x: x\*2

Эквивалентна:

def double(x):

return x \* 2

* Эта функция может иметь любое количество аргументов, но вычисляет и возвращает только одно значение
* Лямбда-функции применимы везде, где требуются объекты-функции
* Вы должны помнить, что  синтаксически лямбда-функция ограничена, позволяет представить всего одно выражение
* Они имеют множество вариантов применения в конкретных областях программирования, наряду  с другими типами выражений, используемых в функциях.

## 2. Различие между обычной функцией и лямбда-функцией

Рассмотрим пример и попробуем понять различие между определением (Def)для обычной функции и lambda**-**функции. Этот код возвращает заданное значение, возведенное в куб:

def defined\_cube(y):  
 return y\*y\*y  
  
  
lambda\_cube = lambda y: y\*y\*y  
print(defined\_cube(2))  
print(lambda\_cube(2))

Вывод:

8

8

Как показано в примере выше, обе представленные функции, defined\_cube() и  lambda\_cube(), ведут себя одинаково, как и предполагалось.

Разберем вышеуказанный пример подробнее:

* **Без использования лямбды:** Здесь обе функции возвращают заданное значение, возведенное в куб. Но при использовании def**,** нам пришлось определить функцию с именем и defined\_cube() дать ей входную величину.  После выполнения нам также понадобилось возвратить результат, из того места, откуда была вызвана функция, и мы сделали это, используя ключевое слово return**.**
* **С применением лямбды:** Определение лямбды не включает оператор return**,** а всегда содержит возвращенное выражение. Мы также можем поместить определение лямбды в любое место, где ожидается функция, и нам не нужно присваивать его переменной. Так выглядят простые лямбда-функции.

## 3. Лямбда-функции и функции высшего порядка

Мы используем лямбда-функцию, когда нам ненадолго требуется безымянная функция.

В Python мы часто используем их как аргумент функции высшего порядка (функции, которая принимает другие функции в качестве аргументов).  Лямбда-функции используют вместе с такими встроенными функциями как filter(), map()**,**reduce() и др.

Давайте рассмотрим еще несколько распространенных вариантов использования лямбда-функций.

### 3.1. Пример с filter()

Функция filter() в Python принимает в качестве аргументов функцию и список .

Функция вызывается со всеми элементами в списке, и в результате возвращается новый список, содержащий элементы, для которых функция результирует в True.

Вот пример использования функции filter() для отбора четных чисел из списка.

my\_list = [1, 3, 4, 6, 10, 11, 15, 12, 14]  
new\_list = list(filter(lambda x: (x%2 == 0) , my\_list))  
print(new\_list)

**Вывод**:

[4, 6, 10, 12, 14]

### 3.2. Пример с map()

Функция map() принимает в качестве аргументов функцию и список.

Функция вызывается со всеми элементами в списке, и в результате возвращается новый список, содержащий элементы, возвращенные данной функцией для каждого исходного элемента.

Ниже пример использования функции map() для удвоения всех элементов списка.

current\_list = [1, 3, 4, 6, 10, 11, 15, 12, 14]  
new\_list = list(map(lambda x: x\*2 , current\_list))  
print(new\_list)

**Вывод:**

[2, 6, 8, 12, 20, 22, 30, 24, 28]

### 3.3. Пример с reduce()

Функция reduce() принимает в качестве аргументов функцию и список. Функция вызывается с помощью лямбда-функции и итерируемого объекта  и возвращается новый уменьшенный результат. Так выполняется повторяющаяся операцию над парами итерируемых объектов. Функция reduce() входит в состав модуля functools**.**

from functools import reduce  
  
  
current\_list = [5, 15, 20, 30, 50, 55, 75, 60, 70]  
summa = reduce((lambda x, y: x + y), current\_list)  
print(summa)

**Вывод:**

380

Здесь результаты предыдущих двух элементов суммируются со  следующим элементом, и это продолжается до конца списка, вот так:

5+15+20+30+50+55+75+60+70

## 4. Лямбда и  списковое включение

В этом примере мы будем использовать лямбда-функцию со списковым включением и лямбда-функцию с циклом for. Мы выведем на экран  таблицу из 10 элементов.

tables = [lambda x = x: x\*10 for x in range(1, 11)]  
for table in tables:  
 print(table())

**Вывод:**

10

20

30

40

50

60

70

80

90

100

## 5. Лямбда и условные операторы

Давайте рассмотрим использование условий if-else в лямбда-функции. Как вы знаете, Python позволяет нам использовать однострочные условия, и  именно их мы можем помещать в лямбда-функцию для обработки возвращаемого результата.

Например, есть две цифры, и вы должны определить, какая из них представляет наибольшее число.

max\_number = lambda a, b: a if a > b else b  
print(max\_number(3, 5))

**Вывод:**

5

Этот метод позволяет вам добавлять условия в лямбда-функции.

## 6. Лямбда и множественные операторы

Лямбда-функции не допускают использования нескольких операторов, однако мы можем создать две лямбда-функции, а затем вызвать вторую лямбда-функцию в качестве параметра для первой функции. Давайте попробуем найти второй по величине элемент, используя лямбду.

current\_list = [[10,6,9],[0, 14, 16, 80],[8, 12, 30, 44]]  
sorted\_list = lambda x: (sorted(i) for i in x)  
second\_largest = lambda x, func: [y[len(y)-2] for y in func(x)]  
result = second\_largest(current\_list, sorted\_list)  
print(result)

**Вывод:**

[9, 16, 30]

В предыдущем примере, мы создали лямбда-функцию, которая сортирует каждый вложенный список в заданном списке. Затем этот список проходит как параметр для второй лямбда-функции, которая возвращает элемент n-2 из отсортированного списка, где  n — длина вложенного списка.



# **Список литературы**

1. https://habr.com/ru/companies/piter/articles/674234/

2. https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html

3. https://ps.readthedocs.io/ru/latest/strings.html

4. https://pythonworld.ru/osnovy/vstroennye-funkcii.html

5. https://tproger.ru/translations/python-built-ins-worth-learning

6. https://sky.pro/media/zachem-ispolzovat-def-main-v-python/

7. https://pythonru.com/uroki/funkcija-main-v-python-dlja-nachinajushhih

8. https://habr.com/ru/articles/525998/