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

Оглавление

[**Введение** 2](#_Toc183002158)

[**1.** **Стандартные функции. Функция main.** 3](#_Toc183002159)

[**2.** **Строковые функции в Python. Преобразование строк** 13](#_Toc183002160)

[**3.** **Лямбда‑функции** 21](#_Toc183002161)

[**Пример лямбда-функции.** 21](#_Toc183002162)

[**Различие между обычной функцией и лямбда-функцией** 22](#_Toc183002163)

[**Лямбда-функции и функции высшего порядка** 23](#_Toc183002164)

[**Лямбда и списковое включение** 25](#_Toc183002165)

[**Лямбда и множественные операторы** 25](#_Toc183002166)

[**Заключение** 27](#_Toc183002167)

[**Список литературы** 27](#_Toc183002168)

**Строковые функции. Лямбда-функции**

**Цель:** формирование знаний в области функционального программирования.

**План лекции:**Введение.1. Стандартные функции. Функция main.  
2. Строковые функции в Python. Преобразование строк.  
3. Лямбда‑функции  
Заключение.Список литературы.

# **Введение**

В процессе разработки программ на языке Python важным аспектом является использование различных функций, которые упрощают и делают код более читаемым и эффективным. В данной лекции мы рассмотрим три ключевых типа функций, которые являются основой работы с Python. Начнем с основных стандартных функций и важнейшей функции программы — main, которая играет важную роль в структуре Python-приложений. Далее обсудим строковые функции, которые позволяют эффективно работать с текстовыми данными, а также методы преобразования строк, необходимые для выполнения различных операций над текстовыми данными. Наконец, мы ознакомимся с лямбда-функциями — важным инструментом для написания компактного и гибкого кода, который может быть использован в различных контекстах.

Эти темы охватывают фундаментальные концепции, которые будут полезны как начинающим программистам, так и более опытным разработчикам, стремящимся улучшить качество и читаемость своего кода. Понимание этих принципов поможет эффективнее использовать возможности Python и ускорит процесс разработки, делая код более элегантным и удобным для поддержки.

# **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. Разберем некоторые стандартные встроенные функции Python.

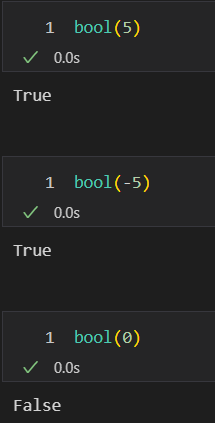
Чтобы разобраться, на какие функции стоит обратить внимание, их следует разделить на группы:

* общеизвестные: почти все новички используют эти функции и довольно часто – мы их уже знаем и используем;
* неочевидные для новичков: об этих функциях полезно знать, но их легко пропустить, когда вы новичок в Python – про них мы поговорим сегодня;
* понадобятся позже: об этих встроенных функциях полезно знать, когда/если они понадобятся, но наперед знать, что понадобится – невозможно;
* 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.

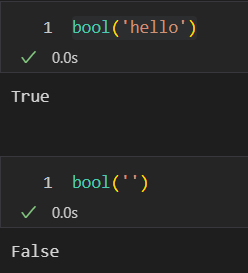
Встроенные функции в первых двух категориях являются основными. Они в конечном итоге будут нужны почти всем начинающим программистам на Python. Встроенные модули в следующих двух категориях являются специализированными, но потребности в них будут варьироваться в зависимости от вашей специализации. Категория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

**Функции, неочевидные для новичков**

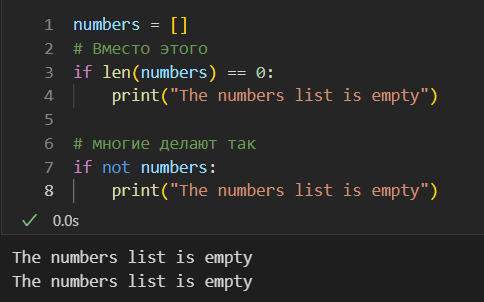
**bool**

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

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



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

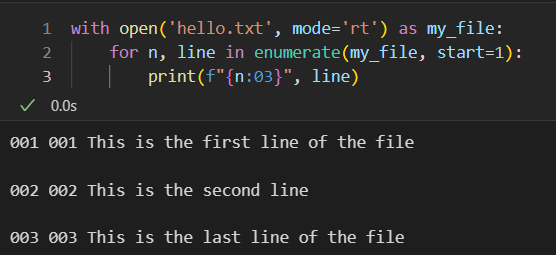


Данная функция используется редко. Но, если нужно привести значение к логическому типу для проверки его истинности, ***bool*** вам необходима.

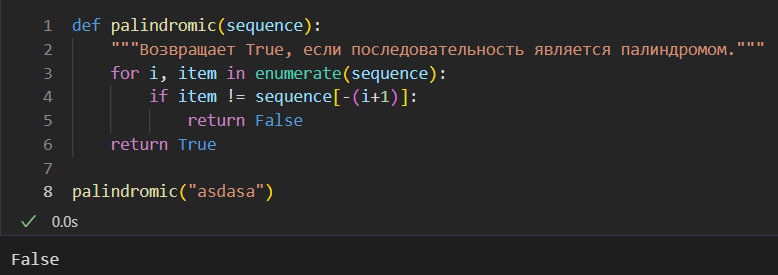
**enumerate**

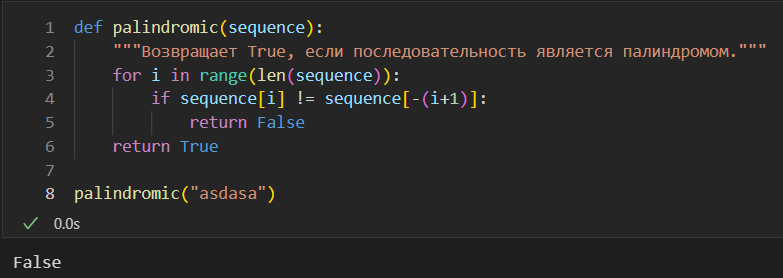
Если нужно в цикле посчитать количество элементов (по одному элементу за раз), эта функция может быть очень полезной. Такая задача может показаться специфической, но она бывает нужна довольно часто.

Например, если нужно отслеживать номер строки в файле.



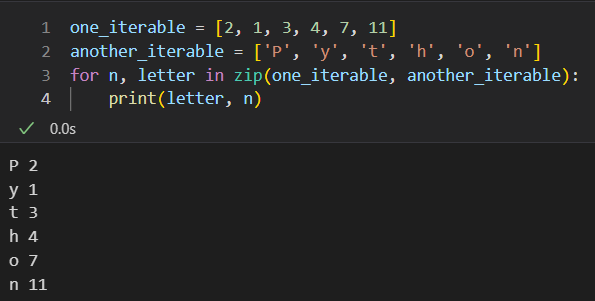
Enumerate также часто используется для отслеживания индекса элементов в последовательности.

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.



**zip**

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

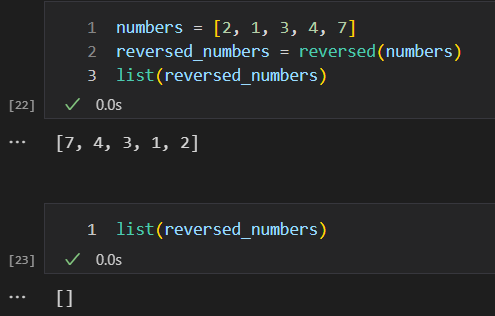


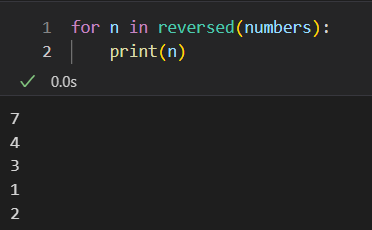
По сравнению с enumerate, последняя функция удобна, когда нужна индексация во время цикла. Если нужно обрабатывать несколько объектов одновременно, zip предпочтительнее enumerate.

**reversed**

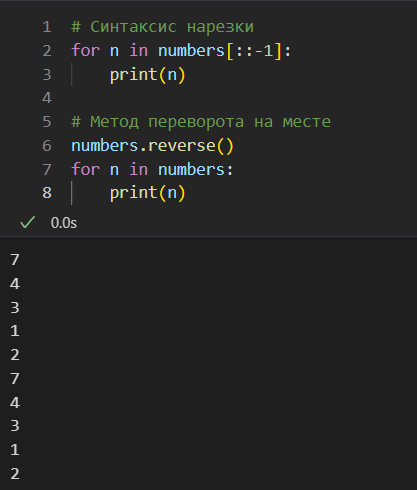
Функция reversed, как enumerate и zip, возвращает итератор.

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



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

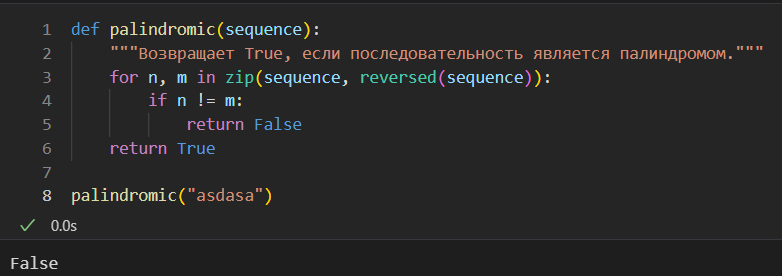
Есть несколько и других способов перевернуть списки в Python.



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.

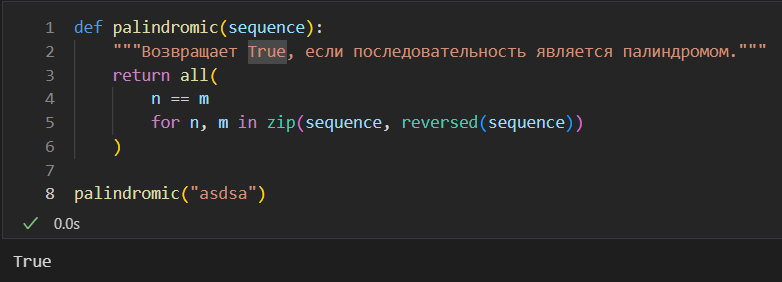
reversed(numbers) (в отличие от numbers [:: - 1]) не создает новый список. Возвращаемый им итератор извлекает следующий элемент в обратном порядке при проходе по циклу. Также синтаксис reversed(numbers) намного более читабелен, чем numbers [:: - 1].

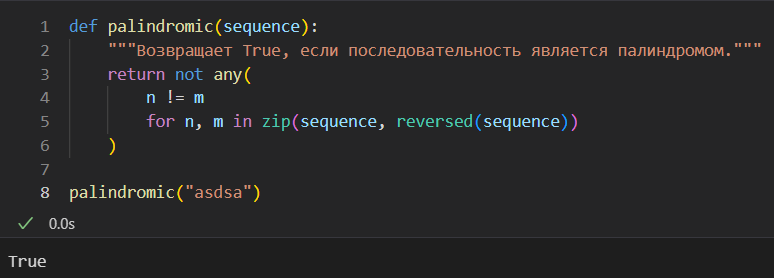
Можно использовать факт, что reversed не копирует список. Если объединить его с функцией zip, можно переписать функцию palindromic (из раздела enumerate), не занимая дополнительной памяти (не копируя объект).

**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.

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



***Main***

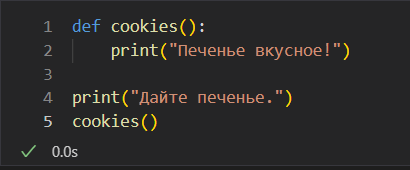
В различных руководствах по Python часто используется функция ***main()***. Но иногда это может быть и лишним.

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:

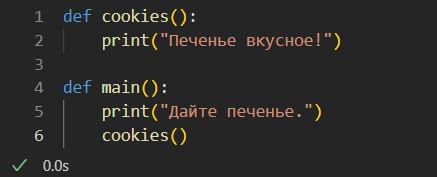


Запустим этот код:

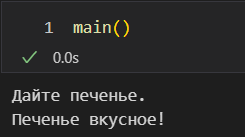


**Как создать функцию main**

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():



Теперь вызовем функцию в конце программы:



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

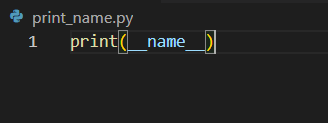
* ***Cookies()*** — функция, которая выводит «Печенье вкусное!».
* Когда программа запускается, определяются две функции: ***cookies()*** и ***main()***.
* Затем функция ***main()*** вызывается.
* «Дайте печенье.» выводится в консоли.
* Затем вызывается функция ***cookies()***, которая выводит в консоль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

Код не только чище, но и логичнее.

**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:



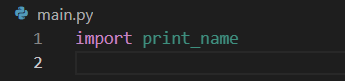
Этот код можно запустить следующим образом:

**$ python print\_name.py**

Он вернет \_\_main\_\_.

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

Копировать



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:

**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.

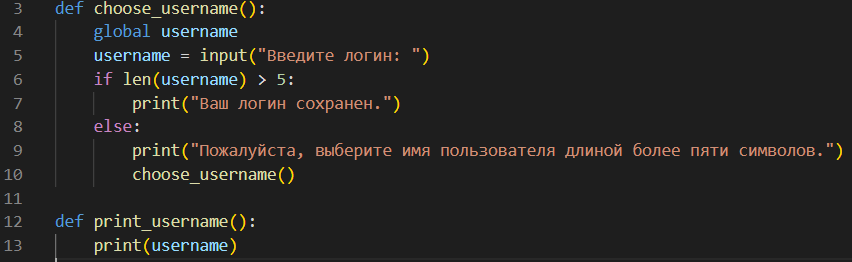
Если же ссылаться на файл как на модуль, то содержимое конструкции if не будет выполнено. Рассмотрим на примере.

**Как использовать \_\_name\_\_ и \_\_main\_\_**

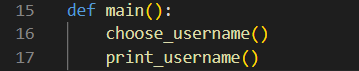
Создадим новый скрипт на Python под названием username.py. В коде будем просить пользователя ввести его имя и проверять, не является ли его длина больше 5. Если символов не больше 5, то попросим ввести имя снова.

Начнем с определения глобальной переменной для хранения имени пользователя:



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: 

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



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.



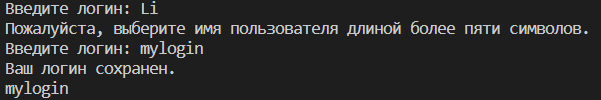
Запустим код:

**$ python username.py**

Он вернет следующее:



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

 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.

**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'

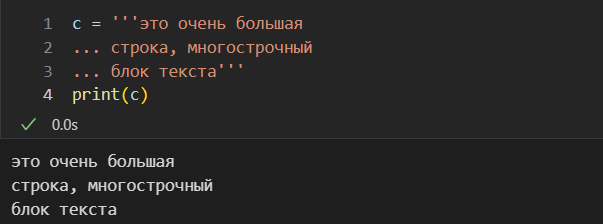
Но, несмотря на назначение, "сырая" строка не может заканчиваться символом обратного слэша. Пути решения:

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

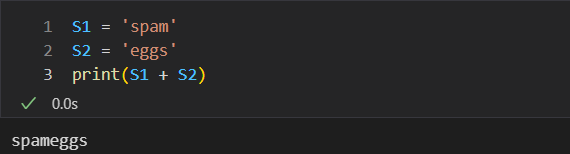
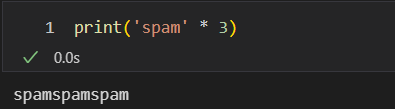
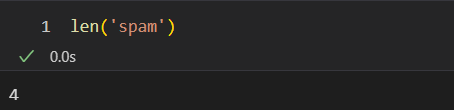
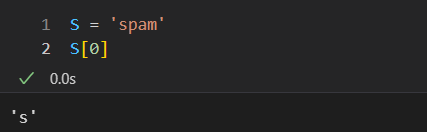
S = r'\n\n' + '\\'

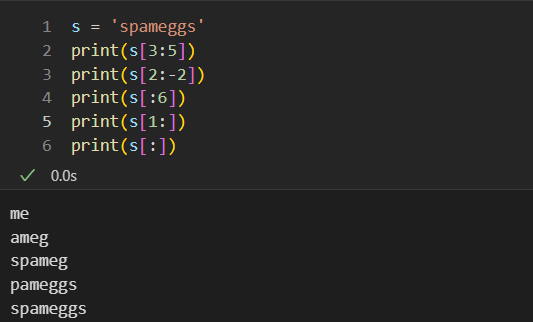
S = '\\n\\n'

**Строки в тройных апострофах или кавычках.**

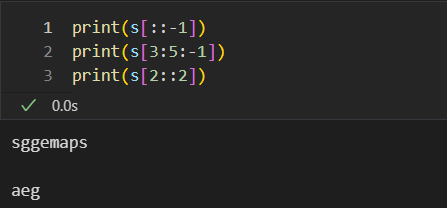
Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд. 

**Базовые операции**

* Конкатенация (сложение) 
* Дублирование строки  
  
* Длина строки (функция len) 
* Доступ по индексу  
  
* Извлечение среза

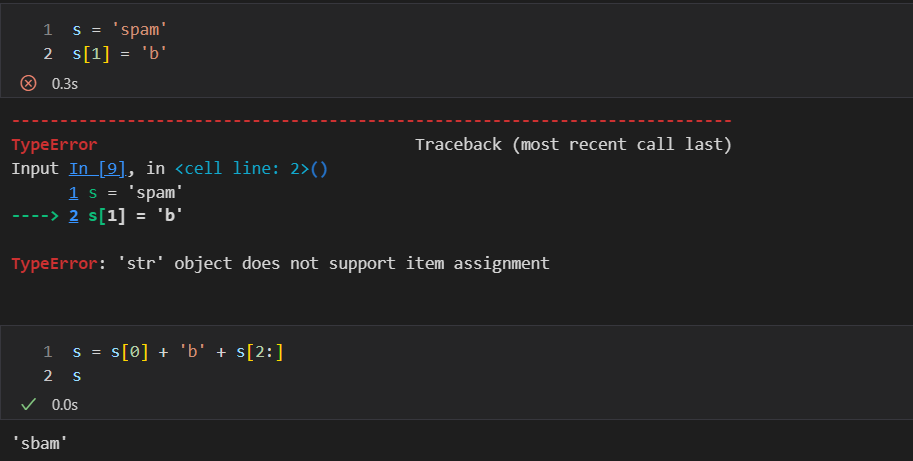
Оператор извлечения среза: [X:Y]. X – начало среза, а Y – окончание; символ с номером Y в срез не входит. По умолчанию первый индекс равен 0, а второй - длине строки.

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



**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.

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

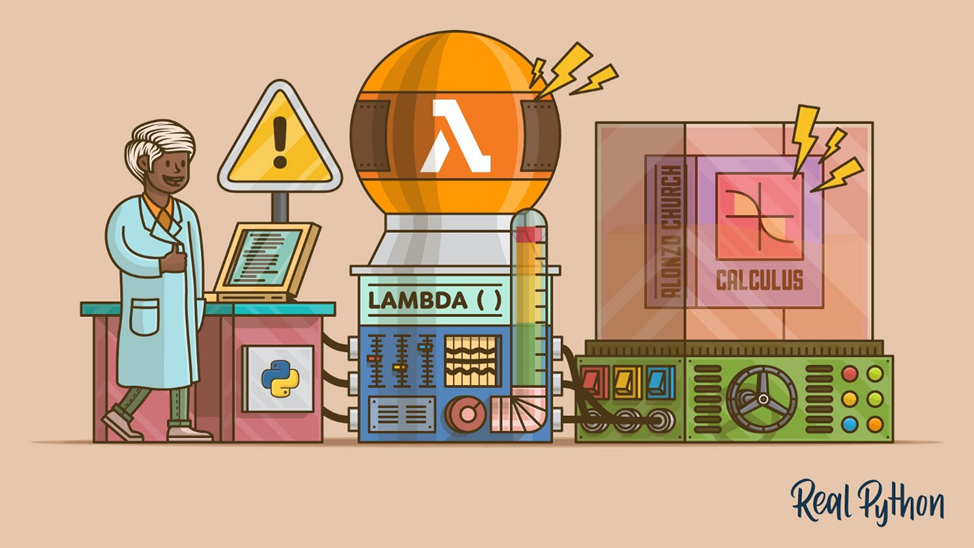
**Таблица "Функции и методы строк"**

|  |  |
| --- | --- |
| 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.

## **Пример лямбда-функции.**

Ниже представлен пример лямбда-функции, удваивающей вводимое значение.

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

**Вывод:**

В вышеуказанном коде ***lambda x: x\*2*** — это лямбда-функция. Здесь ***x*** — это аргумент,а ***x\*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:

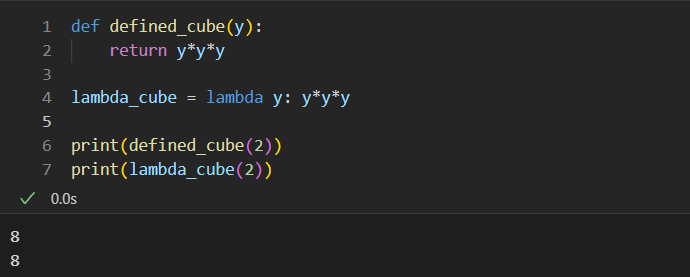
**def double(x):**

**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.

## **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:

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.

## **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.

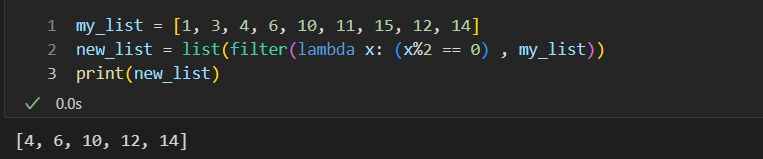
Давайте рассмотрим еще несколько распространенных вариантов использования лямбда-функций.

**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.

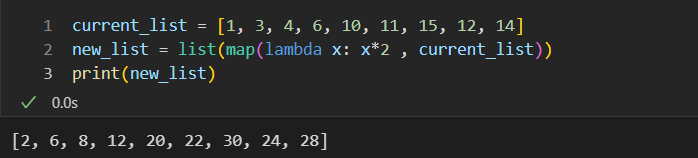
Вот пример использования функции *filter*() для отбора четных чисел из списка.

 **Пример с map()**

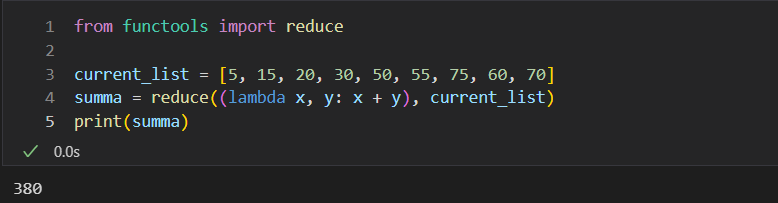
Функция *map*() принимает в качестве аргументов функцию и список.

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.

 **Пример с reduce()**

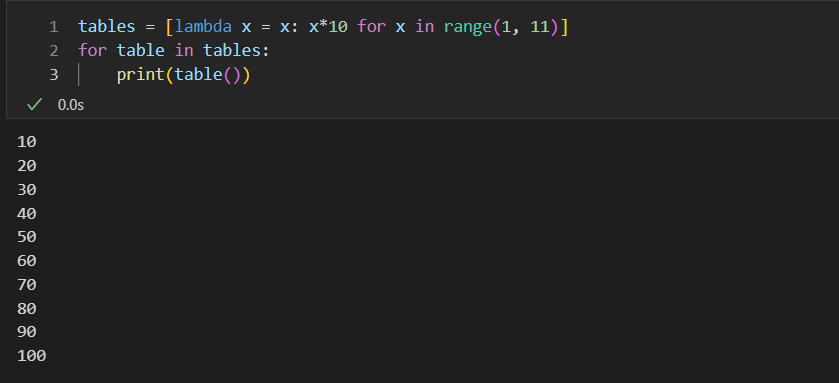
Функция *reduce*() принимает в качестве аргументов функцию и список. Функция вызывается с помощью лямбда-функции и итерируемого объекта  и возвращается новый уменьшенный результат. Так выполняется повторяющаяся операцию над парами итерируемых объектов. Функция *reduce*() входит в состав модуля *functools***.**

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

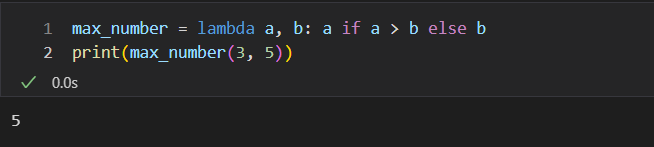
## **Лямбда и списковое включение**

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.

**Лямбда и условные операторы**

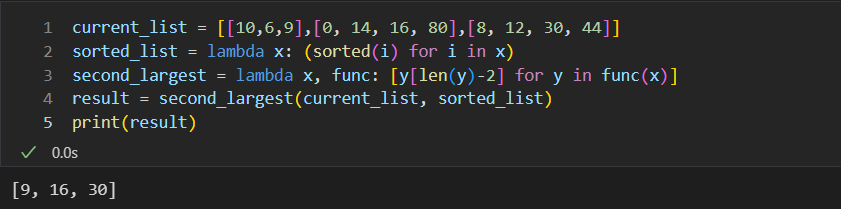
Давайте рассмотрим использование условий *if-else* в лямбда-функции. Как вы знаете, Python позволяет нам использовать однострочные условия, и именно их мы можем помещать в лямбда-функцию для обработки возвращаемого результата.

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

Этот метод позволяет вам добавлять условия в лямбда-функции.

## **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.

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.

# **Заключение**

В заключение, изучив стандартные функции в Python, включая функцию main, мы познакомились с основами организации кода, что позволяет структурировать программу так, чтобы она была удобна для дальнейшей модификации и тестирования. Функции позволяют эффективно управлять потоком исполнения программы и повышают её гибкость. Знание строковых функций и методов преобразования строк является не менее важным для работы с текстовыми данными, так как текстовые манипуляции встречаются в каждом проекте, будь то обработка пользовательского ввода, работа с данными или взаимодействие с внешними API.

Лямбда-функции, в свою очередь, расширяют возможности Python, предоставляя средство для создания кратких и мощных выражений, которые идеально подходят для использования в таких случаях, как сортировка, фильтрация или применение функций в функциональном стиле. Такие функции позволяют написать компактный, но при этом читаемый код, что особенно важно при работе с большими объемами данных или сложными алгоритмами. Таким образом, знания, полученные в ходе этой лекции, являются необходимыми для того, чтобы стать более эффективным и уверенным пользователем Python.

# **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/