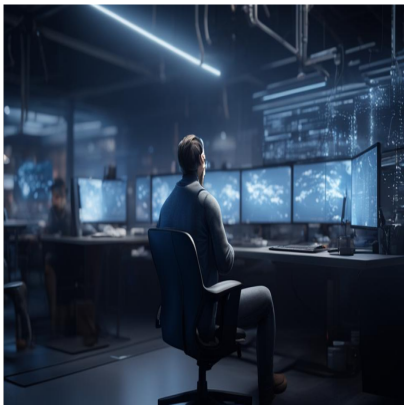


Как использовать лямбда-функции в Python

Описание

В этом уроке вы узнаете все о том, как использовать лямбда-функции в Python – от синтаксиса для определения лямбда-функций до различных вариантов использования с примерами кода. В Python лямбды – это *анонимные* функции, которые имеют лаконичный синтаксис и могут использоваться вместе с другими полезными встроенными функциями. К концу этого урока вы узнаете, как определять лямбда-функции и когда их следует использовать вместо обычных функций Python. Давайте начнем!

Лямбда-функция Python: Синтаксис и примеры

Вот общий синтаксис для определения лямбда-функции в Python:

```
lambda parameter(s):return value
```

В приведенном выше общем синтаксисе:

- **lambda** – это ключевое слово, которое нужно использовать для определения лямбда-функции, за которым следует один или несколько **параметров**, которые должна принимать функция.
- Параметры и **возвращаемое значение** разделяются двоеточием.

При определении лямбда-функции необходимо убедиться, что возвращаемое значение вычисляется путем оценки выражения, состоящего из одной строки кода. Вы поймете это лучше, когда мы разберем примеры.

Примеры лямбда-функций Python

Лучший способ понять лямбда-функции – начать с переписывания обычных функций Python в виде лямбда-функций.

#1. Рассмотрим следующую функцию `square()`, которая принимает в качестве аргумента число `num` и возвращает квадрат этого числа.

```
def square(num): return  num*num
```

Вы можете вызвать функцию с аргументами и проверить, что она работает правильно.

```
>>> square(9)
81
>>> square(12)
144
```

Вы можете присвоить это лямбда-выражение переменной с именем, скажем, `square1`, чтобы сделать определение функции более кратким: `square1 = lambda num:  num*num`, а затем вызвать функцию `square1` с любым числом в качестве аргумента. Однако мы знаем, что лямбды – это анонимные функции, поэтому не стоит присваивать их переменной. Для функции `square()` параметром является `num`, а возвращаемым значением – `num*num`. После того как мы их определили, мы можем вставить их в лямбда-выражение и вызвать его с аргументом, как показано на рисунке:

```
>>> (lambda num:  num*num)(2)4
```

Это концепция Immediately Invoked Function Expression, когда мы вызываем функцию сразу после ее определения.

#2. Далее перепишем еще одну простую функцию `add()`, которая берет числа `num1` и `num2` и возвращает их сумму, `num1 + num2`.

```
def add(num1,num2): return num1 + num2
```

Вызовем функцию add() с двумя числами в качестве аргументов:

```
>>> add(4,3)7>>> add(12,5)17>>> add(12,6)18
```

В данном случае num1 и num2 – это два параметра, а возвращаемое значение – num1 + num2.

```
>>> (lambda num1, num2: num1 + num2)(3,7)10
```

Функции Python также могут принимать значения параметров по умолчанию. Давайте изменим определение функции add() и установим значение параметра num2 по умолчанию равным 10.

```
def add(num1, num2=10): return num1 + num2
```

В следующих вызовах функций:

- В первом вызове функции значение num1 равно 1, а значение num2 равно 3. Когда вы передаете значение num2 в вызове функции, это значение используется; функция возвращает 4.
- Однако если вы передаете только один аргумент(num1 равен 7), то для num2 используется значение по умолчанию 10; функция возвращает 17.

```
>>> add(1,3)4>>> add(7)17
```

При написании функций, принимающих значения по умолчанию для определенных параметров в виде лямбда-выражений, вы можете указать значение по умолчанию при определении параметров.

```
>>> (lambda num1, num2 = 10: num1 + num2)(1)11
```

Когда следует использовать лямбда-функции в Python?

Теперь, когда вы изучили основы лямбда-функций в Python, вот несколько примеров их использования:

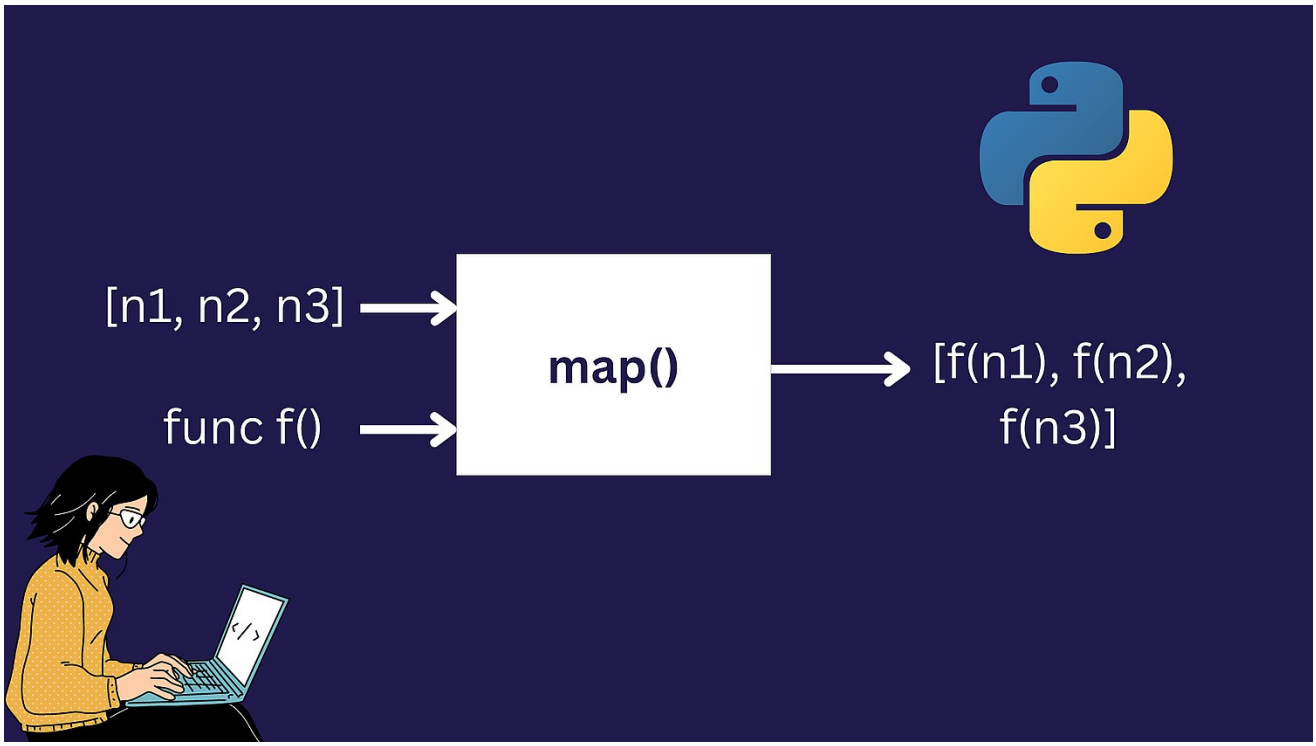
- Когда у вас есть функция, возвращаемое выражение которой состоит из одной строки кода, и вам не нужно ссылаться на эту функцию в другом месте того же модуля, вы можете использовать лямбда-функции. Мы также написали несколько примеров, чтобы понять это.

- Вы можете использовать лямбда-функции при работе со встроенными функциями, такими как map(), filter() и reduce().
- Лямбда-функции могут быть полезны при сортировке структур данных Python, таких как списки и словари.

Как использовать лямбду в Python со встроенными функциями

Использование лямбды с Map()

Функция `map()` принимает итерируемую переменную и функцию и применяет функцию к каждому элементу итерируемой переменной, как показано на рисунке:



Создадим список `nums` и с помощью функции `map()` создадим новый список, содержащий квадрат каждого числа в списке `nums`. Обратите внимание на использование лямбда-функции для определения операции возведения в квадрат.

```
>>> nums = [4,5,6,9]>>> list(map(lambda num:num*num,nums))[16, 25, 36, 81]
```

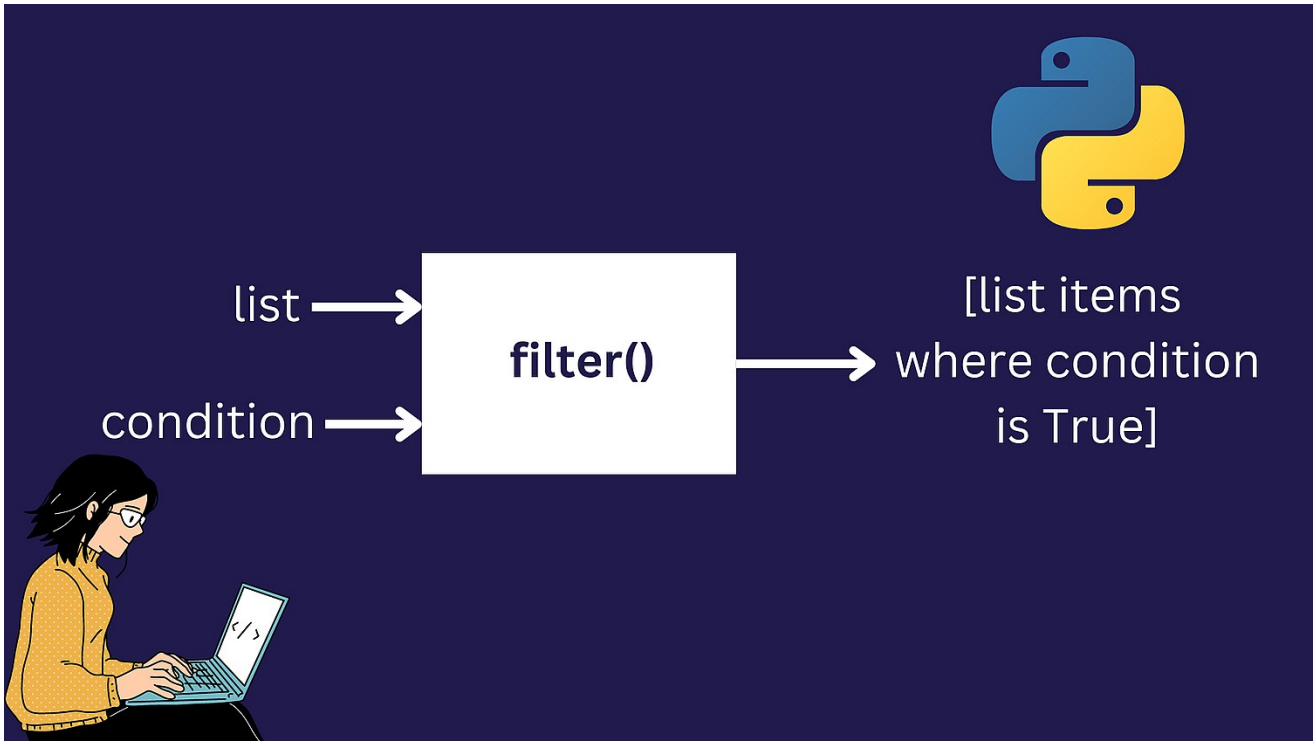
Поскольку функция `map()` возвращает объект map, мы должны преобразовать его в список.

Использование лямбды с Filter()

Давайте определим `nums`– список чисел:

```
>>> nums = [4,5,6,9]
```

Предположим, вы хотите отфильтровать этот список и сохранить только нечетные номера. Вы можете использовать встроенную в Python функцию `filter()`. Функция `filter()` принимает **условие** и **итерируемую переменную**: `filter(condition, iterable)`. Результат содержит только те элементы исходной итерируемой таблицы, которые удовлетворяют условию. Вы можете преобразовать возвращаемый объект в итерабельную переменную Python, например список.



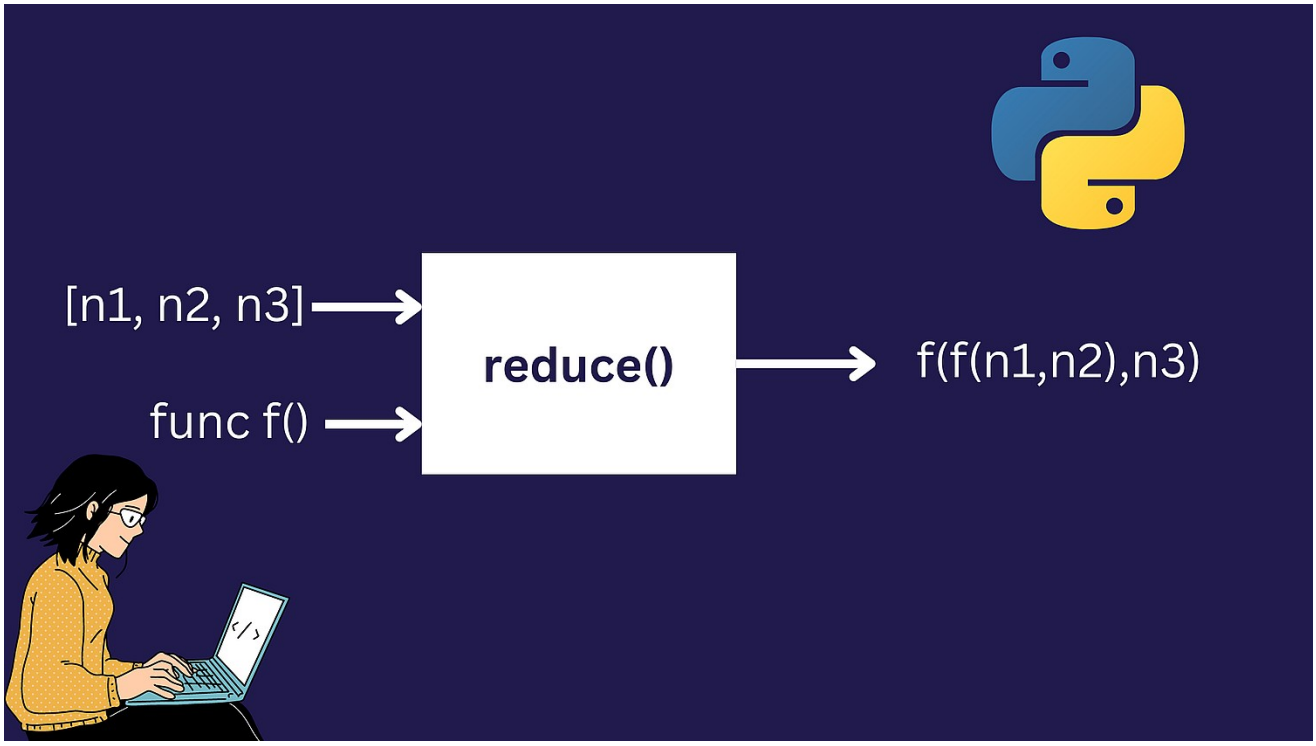
Чтобы отсеять все четные числа, мы оставим только нечетные. Поэтому лямбда-выражение должно иметь вид `lambda num: num%2!=0`. Количество `num%2` – это остаток при делении `num` на 2.

- `num%2!=0` – это `True`, когда `num` нечетно, и
- `num%2!=0` – это `False`, если `num` четное.

```
>>> nums = [4,5,6,9]>>> list(filter(lambda num:num%2!=0,nums))[5, 9]
```

Использование лямбды с Reduce()

Функция `reduce()` принимает итерируемую переменную и функцию. Она уменьшает итерабельную таблицу, применяя функцию итеративно к элементам итерабельной таблицы.



Чтобы использовать функцию `reduce()`, вам придется импортировать ее из встроенного модуля Python `functools`:

```
>>> from functools import reduce
```

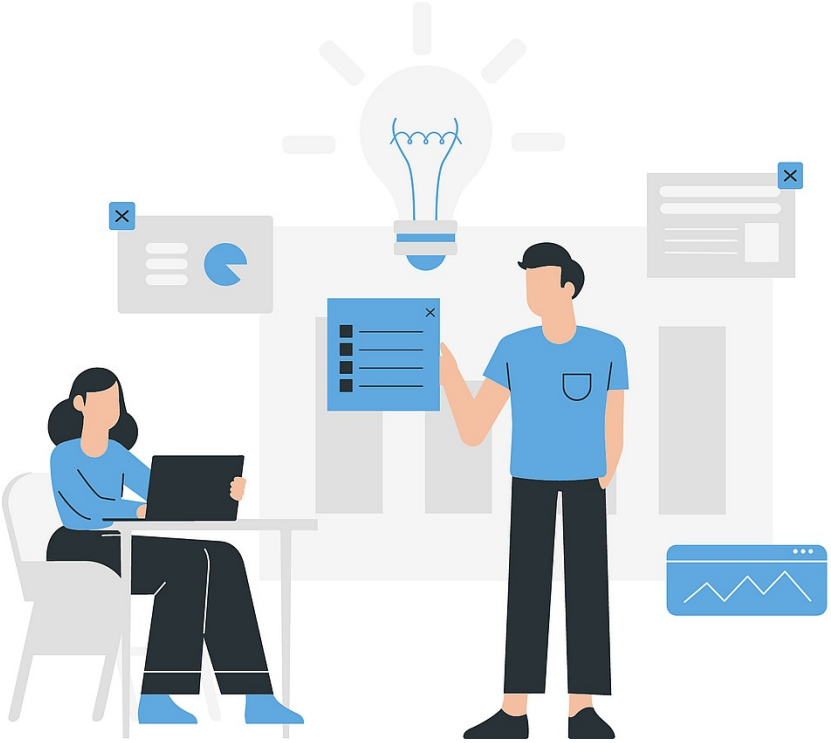
Давайте воспользуемся функцией reduce(), чтобы вычислить сумму всех чисел в списке nums. В качестве функции уменьшения суммы определим лямбда-выражение: `lambda num1,num2:num1+num2`. Операция уменьшения будет выглядеть так: f(f(f(4,5),6),9) = f(f(9,6),9) = f(15,9) = 24. Здесь f – операция суммирования по двум элементам списка, определяемая лямбда-функцией.

```
>>> from functools import reduce>>> nums = [4,5,6,9]>>> reduce(lambda num1,num
```

Лямбда-функции Python для настройки сортировки

Помимо использования лямбда-функций со встроенными функциями Python, такими как `map()`, `filter()` и `reduce()`, вы также можете использовать их для настройки

встроенных функций и методов, используемых для сортировки.



Сортировка списков в Python

При работе со списками Python вам часто придется сортировать их на основе определенных критериев сортировки. Чтобы отсортировать списки Python на месте, вы можете использовать встроенный метод `sort()`. Если вам нужна отсортированная копия списка, вы можете использовать функцию `sorted()`.

> Синтаксис для использования функции Python `sorted()` – `sorted(iterable, key=...,reverse= True | False)`.
>
> – Параметр `key` используется для настройки сортировки.
> – Параметр `reverse` может быть установлен в `True` или `False`; по умолчанию используется значение `False`.

При сортировке списков чисел и строк по умолчанию используется сортировка по возрастанию и в алфавитном порядке соответственно. Однако иногда вам может понадобиться задать какой-либо пользовательский критерий для сортировки. Рассмотрим следующий список `fruits`. Предположим, вы хотите получить отсортированную копию этого списка. Вы должны отсортировать строки по количеству вхождений в них буквы ‘p’ – в порядке убывания.

```
>>> fruits = ['apple','pineapple','grapes','mango']
```

Пришло время использовать необязательный параметр key. В Python строка является итерируемой, и чтобы получить количество вхождений символа в нее, можно использовать встроенный метод `.count()`. Поэтому мы зададим ???? в виде `lambda x:x.count('p')`, чтобы сортировка происходила по количеству повторений символа ‘p’ в строке.

```
>>> fruits = ['apple','pineapple','grapes','mango']>>> sorted(fruits,key=lambd
```

В данном примере:

- ???? для сортировки – это количество вхождений символа ‘p’, и он задается в виде лямбда-выражения.
- Поскольку мы установили параметр `reverse` в `True`, сортировка происходит в порядке убывания количества вхождений ‘p’.

В списке `fruits` строка ‘pineapple’ содержит 3 вхождения ‘p’, а строки ‘apple’, ‘grapes’ и ‘mango’ содержат 2, 1 и 0 вхождений ‘p’ соответственно.

Понимание стабильной сортировки

Рассмотрим другой пример. Для того же критерия сортировки мы переопределили список ???????. Здесь ‘p’ встречается в строках ‘яблоко’ и ‘виноград’ дважды и один раз соответственно. И никогда не встречается в строках ‘mango’ и ‘melon’.

```
>>> fruits = ['mango','apple','melon','grapes']>>> sorted(fruits,key=lambda x:
```

В списке вывода ‘mango’ стоит перед ‘melon’, хотя оба они не содержат символа ‘p’. Но почему так происходит? Функция `sorted()` выполняет стабильную сортировку, поэтому, когда количество символов ‘p’ одинаково для двух строк, порядок элементов в исходном списке ??????? сохраняется.

> В качестве быстрого упражнения поменяйте местами “манго” и “дыня” в списке ???????, отсортируйте список по тому же критерию и понаблюдайте за результатом.

Сортировка словаря Python

Вы также можете использовать лямбды при сортировке словарей Python. Рассмотрим следующий словарь `price_dict`, содержащий товары и их цены.

```
>>> price_dict = {... '??????':10,... '???':15,... '????':7,... '???????':3...
```

Чтобы получить пары ключ-значение из словаря в виде списка кортежей, можно воспользоваться встроенным методом словаря `.items()`:

```
>>> price_dict_items = price_dict.items()dict_items([('??????', 10), ('???', 1
```

> В Python все итерируемые объекты: списки, кортежи, строки и другие – имеют нулевую индексацию. То есть первый элемент имеет индекс 0, второй – индекс 1 и так далее.

Мы хотим сортировать по значению, которое является ценой каждого элемента в словаре. В каждом кортеже в списке `price_dict_items` ценой является элемент с индексом 1. Поэтому мы задаем ???? в виде `lambda x:x[1]`, так как он будет использовать элемент с индексом 1, цену, для сортировки словаря.

```
>>> dict(sorted(price_dict_items,key=lambda x:x[1])){'???????': 3, '????': 7,
```

В выводе словарные статьи были отсортированы в порядке возрастания цен: начиная с “Конфет”, стоимостью 3 единицы, и заканчивая “Медом”, стоимостью 15 единиц.

Подведение итогов

Вот и все! Вы узнали, как определять лямбда-функции и эффективно использовать их вместе с другими встроенными функциями Python. Вот краткое изложение основных выводов:

- В Python лямбды – это **анонимные функции**, которые могут принимать несколько аргументов и возвращать значение; выражение, которое должно быть вычислено для создания возвращаемого значения, должно состоять из

одной строки кода. Их можно использовать для того, чтобы сделать небольшие определения функций более краткими.

- Для определения лямбда-функции можно использовать синтаксис: **lambda parameter(s): return value**.
- Среди важных примеров использования – применение их в функциях `map()`, `filter()` и `reduce()`, а также в качестве ключевого параметра для настройки сортировки итерационных таблиц Python.

Дата Создания
18.04.2024