Lambda Functions In Python

In Python, a lambda function (also called an anonymous function) is a small, anonymous function that can be defined in a single line of code without a name. It is useful when we need a simple function that we don't want to define explicitly using the def keyword. The basic syntax for a lambda function in Python is:

lambda arguments: expression Here, arguments refer to the input arguments for the function and expression is a single expression that gets evaluated and returned as the result of the function. The result of the expression is automatically returned by the lambda function, so there's no need to use the return statement.

1) Lambda functions are defined using the keyword lambda, followed by the arguments (separated by commas) and a colon, and then the expression to be evaluated.

2) Lambda functions can take any number of arguments, including optional arguments with default values.

3) The expression in a lambda function can only be a single line of code, and it is evaluated and returned automatically.

4) Lambda functions can be assigned to variables, passed as arguments to other functions, and even returned by functions.

5) Lambda functions are often used in conjunction with higher-order functions like map(), filter(), and reduce() to perform simple operations on collections of data.

6) Lambda functions are not always the most readable or maintainable way to write code, and should be used judiciously.

1. A lambda function that takes in two arguments and returns their sum:

In [90]:
```
sum_lambda = lambda a, b: a + b
print(sum_lambda(5, 10))
```

```
15
```

1. A lambda function that takes in a list and returns the length of the list:

In [91]:
```
length_lambda = lambda lst: len(lst)
fruits = ['apple', 'banana', 'orange']
print(length_lambda(fruits))
```

```
3
```

1. A lambda function that takes in a string and returns the uppercase version of the string:

In [92]:
```
uppercase_lambda = lambda s: s.upper()
print(uppercase_lambda('hello world'))
```

```
HELLO WORLD
```

1. A lambda function that takes in a dictionary and returns the keys of the dictionary:

In [93]:
```
keys_lambda = lambda d: list(d.keys())
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(keys_lambda(my_dict))
```

```
['a', 'b', 'c']
```

1. A lambda function that takes in a number and returns whether the number is even or odd:

In [94]:
```
even_odd_lambda = lambda num: 'Even' if num % 2 == 0 else 'Odd'
print(even_odd_lambda(7))
```

```
Odd
```

In [57]:
```
multiply = lambda x,y: x*y
```

In [58]:
```
multiply(11,12)
```

Out[58]:
```
132
```

In [59]:
```
power = lambda x,y: x**y
```

In [60]:
```
power(2,3)
```

Out[60]:
```
8
```

In [4]:
```
x = lambda x:x**2
```

In [5]:
```
x(9)
```

Out[5]:
```
81
```

In [6]:
```
a = lambda x,y:x+y
```

In [7]:
```
a(4,5)
```

Out[7]:
```
9
```

In [ ]:
```
# Differenece
1. Lambda has no return value
2. One line
3. Not used for code reusability
4. No name in lambda like def power
```

In [8]:
```
type(a)
```

Out[8]:
```
function
```

In [ ]:
```
# why lambda function ?
# Along with Higher order functions
```

In [9]:
```
b = lambda x:x[0]=='a'
b('apple')
```

Out[9]:
```
True
```

In [10]:
```
b('banana')
```

Out[10]:
```
False
```

In [11]:
```
# Examine Even Or Odd
# If Else

b = lambda x:"Even" if x%2==0 else "Odd"
b(3)
```

Out[11]:
```
'Odd'
```

In [12]:
```
b(2)
```

Out[12]:
```
'Even'
```

Higher order function

In [15]:
```
def return_sum(func,L):
    result=0
    for i in L:
        if func(i):
            result = result + i
    return result

L = [11,14,21,23,56,78,45,29,28]

x = lambda x:x%2==0
y = lambda x:x%2!=0
z = lambda x:x%3==0

print(return_sum(x,L))
print(return_sum(y,L))
print(return_sum(z,L))
```

```
176
129
144
```

Map Function

The map() function in Python is a built-in higher-order function that takes a function and an iterable as arguments, and returns a new iterable with the function applied to each element of the original iterable. Here are some examples of how to use the map() function in Python:

In [95]:
```
numbers = [1, 2, 3, 4, 5]
squares = map(lambda x: x ** 2, numbers)

print(list(squares))
```

```
[1, 4, 9, 16, 25]
```

In [17]:
```
# Map
L = [1,2,3,4,5,6,7]
list(map(lambda x:x**2,L))
```

Out[17]:
```
[1, 4, 9, 16, 25, 36, 49]
```

In [19]:
```
list(map(lambda x : x%2 == 0,L))
```

Out[19]:
```
[False, True, False, True, False, True, False]
```

In [25]:
```
students = [
{
    "name":"Akash pawar",
    "father name":"Rama pawar",
    "Address":"123 Hill Street"
},{
    "name":"Prem Rajput",
    "father name":"Dilip Rajput",
    "Address":"456 Hill Street"
},{
    "name":"Sanket Bendre",
    "father name":"Vilas Bendre",
    "Address":"789 Hill Street"
}
]
```

In [26]:
```
list(map(lambda student:student['name'],students))
```

Out[26]:
```
['Akash pawar', 'Prem Rajput', 'Sanket Bendre']
```

In [27]:
```
list(map(lambda student:student['Address'],students))
```

Out[27]:
```
['123 Hill Street', '456 Hill Street', '789 Hill Street']
```

In [64]:
```
strings = ["My","friends"]
```

In [69]:
```
cap = list(map(lambda x: str.upper(x),strings))
```

In [71]:
```
cap
```

Out[71]:
```
['MY', 'FRIENDS']
```

Filter Function

The filter() function in Python is a built-in higher-order function that takes a function and an iterable as arguments, and returns a new iterable with only the elements from the original iterable for which the function returns True. Here are some examples of how to use the filter() function in Python.

In [96]:
```
numbers = [1, 2, 3, 4, 5]
evens = filter(lambda x: x % 2 == 0, numbers)

print(list(evens))
```

```
[2, 4]
```

In [28]:
```
# Filter
L = [1,2,3,4,5,6,7]
```

In [29]:
```
list(filter(lambda x:x>4,L))
```

Out[29]:
```
[5, 6, 7]
```

In [97]:
```
fruits = ['apple', 'banana', 'orange', 'kiwi']
long_fruits = filter(lambda x: len(x) > 5, fruits)

print(list(long_fruits))
```

```
['banana', 'orange']
```

In [98]:
```
people = [('Alice', 25), ('Bob', 30), ('Charlie', 35), ('Dave', 40)]
seniors = filter(lambda x: x[1] >= 35, people)

print(list(seniors))
```

```
[('Charlie', 35), ('Dave', 40)]
```

In [99]:
```
scores = {'Alice': 80, 'Bob': 70, 'Charlie': 90, 'Dave': 75}
high_scores = filter(lambda x: x[1] >= 80, scores.items())

print(dict(high_scores))
```

```
{'Alice': 80, 'Charlie': 90}
```

In [31]:
```
fruits = ['Apple','Orange','Mango','Guava']
list(filter(lambda fruit: 'e' in fruit, fruits))
```

Out[31]:
```
['Apple', 'Orange']
```

Reduce Function

In [32]:
```
import functools
```

In [33]:
```
L = [1,2,3,4,5,6,7]
L
```

Out[33]:
```
[1, 2, 3, 4, 5, 6, 7]
```

In [34]:
```
# Sum list
functools.reduce(lambda x,y:x+y,L)
```

Out[34]:
```
28
```

In [35]:
```
L1 = [12,34,56,11,21,58]
L1
```

Out[35]:
```
[12, 34, 56, 11, 21, 58]
```

In [36]:
```
# Max value
functools.reduce(lambda x,y:x if x>y else y, L1)
```

Out[36]:
```
58
```

In [37]:
```
# Min value
functools.reduce(lambda x,y:x if x<y else y, L1)
```

Out[37]:
```
11
```

Lambda on dictionaries

In [77]:
```
sales = [{'country':"India",'sale':150.5}, {'country':"Japan",'sale':200.2},{'country':"Eu
```

In [78]:
```
country_key = map(lambda x: x['country'],sales)
```

In [79]:
```
list(country_key)
```

Out[79]:
```
['India', 'Japan', 'Europe']
```

In [80]:
```
country_values = map(lambda x: x['sale'],sales)
```

In [81]:
```
list(country_values)
```

Out[81]:
```
[150.5, 200.2, 300.12]
```

In [ ]:
```
# Using filter on dictionaries along with lambda
```

In [83]:
```
India_sales =filter(lambda x : x['country'] == "India",sales)
```

In [84]:
```
list(India_sales)
```

Out[84]:
```
[{'country': 'India', 'sale': 150.5}]
```

In [85]:
```
high_sales = filter(lambda x : x['sale'] > 200, sales)
```

In [86]:
```
list(high_sales)
```

Out[86]:
```
[{'country': 'Japan', 'sale': 200.2}, {'country': 'Europe', 'sale': 300.12}]
```

Processing multiple list with map & lambda function

In [87]:
```
list1 = [20,24,23,64,54,59]
list2 = [12,34,56,62,23,45]
```

In [88]:
```
list_addition = map(lambda x,y: x+y, list1, list2)
```

In [89]:
```
list(list_addition)
```

Out[89]:
```
[32, 58, 79, 126, 77, 104]
```

List Comprehension

In [38]:
```
# List Comprehension
L = [1,2,3,4,5,6,7]
```

In [42]:
```
L1 = [item * 2 for item in L]
L1
```

Out[42]:
```
[2, 4, 6, 8, 10, 12, 14]
```

In [41]:
```
L2 = [i**2 for i in range(10)]
L2
```

Out[41]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [44]:
```
L3 = [i**2 for i in range(10) if i%2!=0]
L3
```

Out[44]:
```
[1, 9, 25, 49, 81]
```

In [45]:
```
fruits
```

Out[45]:
```
['Apple', 'Orange', 'Mango', 'Guava']
```

In [47]:
```
L4 = [fruit for fruit in fruits if fruit[0]=='O']
L4
```

Out[47]:
```
['Orange']
```

Dictionary Comprehension

In [49]:
```
D = {"Name":"Akshay","Gender":"Male","Age":30}
```

In [50]:
```
D.items()
```

Out[50]:
```
dict_items([('Name', 'Akshay'), ('Gender', 'Male'), ('Age', 30)])
```

In [52]:
```
D1 = {key:value for key,value in D.items() if len(key) > 3}
D1
```

Out[52]:
```
{'Name': 'Akshay', 'Gender': 'Male'}
```

In [54]:
```
D2 = {item:item**2 for item in L}
D2
```

Out[54]:
```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}
```

In [56]:
```
D2 = {item:item**2 for item in L if item%2==0}
D2
```

Out[56]:
```
{2: 4, 4: 16, 6: 36}
```

Lambda Function In Python

In [21]:
```
f=lambda x,y:x+y
f
```

Out[21]:
```
<function __main__.<lambda>(x, y)>
```

In [22]:
```
f(5,6)
```

Out[22]:
```
11
```

In [23]:
```
## simnples examples
multiply=lambda x,y:x*y
multiply(3,4)
```

Out[23]:
```
12
```

In [24]:
```
## Return the length of a string
string_length=lambda s:len(s)
string_length("Dipak Mani")
```

Out[24]:
```
10
```

In [25]:
```
## Convert a list of integers to their corresponding square values:
numbers=[1,2,3,4,5,6]
squares=list(map(lambda x:x**2,numbers))
print(squares)
```

```
[1, 4, 9, 16, 25, 36]
```

In [26]:
```
## Filter out even numbers from a list:

numbers=[1,2,3,4,5,6]
list(filter(lambda x:x%2==0,numbers))
```

Out[26]:
```
[2, 4, 6]
```

In [27]:
```
f=lambda x:x%2==0
f(3)
```

Out[27]:
```
False
```

In [28]:
```
## Sort a list of strings based on their alphabetical characters and  length:
fruits = ['apple', 'banana', 'cherry', 'date', 'elderberry']
```

In [29]:
```
sorted(fruits,key=lambda x:len(x))
```

Out[29]:
```
['date', 'apple', 'banana', 'cherry', 'elderberry']
```

In [30]:
```
## complex examples
## Sorting a list of dictionaries based on a specific key
```

In [31]:
```
people = [
    {'name': 'Alice', 'age': 25, 'occupation': 'Engineer'},
    {'name': 'Bob', 'age': 30, 'occupation': 'Manager'},
    {'name': 'Charlie', 'age': 22, 'occupation': 'Intern'},
    {'name': 'Dave', 'age': 27, 'occupation': 'Designer'},
]
```

In [32]:
```
sorted(people,key=lambda x:(x['age']))
```

Out[32]:
```
[{'name': 'Charlie', 'age': 22, 'occupation': 'Intern'},
 {'name': 'Alice', 'age': 25, 'occupation': 'Engineer'},
 {'name': 'Dave', 'age': 27, 'occupation': 'Designer'},
 {'name': 'Bob', 'age': 30, 'occupation': 'Manager'}]
```

In [33]:
```
## Finding the maximum value in a dictionary
data = {'a': 10, 'b': 20, 'c': 5, 'd': 15}
max(data,key=lambda x:data[x])
```

Out[33]:
```
'b'
```

In [34]:
```
## Grouping a list of strings based on their first letter
```

In [35]:
```
from itertools import groupby

words = ['apple', 'banana', 'cherry', 'date', 'elderberry', 'fig']

groups = groupby(sorted(words), key=lambda x: x[0])

for key, group in groups:
    , list(group))
```

```
a ['apple']
b ['banana']
c ['cherry']
d ['date']
e ['elderberry']
f ['fig']
```

1. A lambda function is an anonymous function, meaning it has no name.

In [11]:
```
square = lambda x: x ** 2
print(square(3))
```

```
9
```

1. The syntax for defining a lambda function is lambda arguments: expression.

In [12]:
```
add = lambda x, y: x + y
print(add(2, 3))
```

```
5
```

1. The arguments are a comma-separated list of arguments that the function can take.

In [13]:
```
concat = lambda a, b, c: a + b + c
print(concat('hello', 'world', '!'))
```

```
helloworld!
```

1. The expression is a single expression that is executed when the function is called, and the result of the expression is returned as the result of the function.

In [14]:
```
is_even = lambda x: x % 2 == 0
print(is_even(4))
```

```
True
```

1. Lambda functions are often used for short, one-off functions that do not need to be defined with a separate def statement.

In [15]:
```
evens = list(filter(lambda x: x % 2 == 0, range(10)))
print(evens)
```

```
[0, 2, 4, 6, 8]
```

1. Lambda functions can take any number of arguments but can only have one expression.

In [16]:
```
full_name = lambda first, last: f'{first.title()} {last.title()}'
print(full_name('john', 'doe'))
```

```
John Doe
```

1. Lambda functions are often used in conjunction with other Python functions like map(), filter(), and reduce()

In [18]:

```
from functools import reduce
nums = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x ** 2, nums))
evens = list(filter(lambda x: x % 2 == 0, nums))
sum = reduce(lambda x, y: x + y, nums)
print(squares)  # Output: [1, 4, 9, 16, 25]
print(evens)  # Output: [2, 4]
print(sum)  # Output: 15
```

```
[1, 4, 9, 16, 25]
[2, 4]
15
```

1. Lambda functions can be used to create closures, which are functions that remember the values of the variables in the enclosing scope even when they are executed in a different scope.

In [19]:

```
def make_adder(n):
    return lambda x: x + n

add_5 = make_adder(5)
result = add_5(10)  # result is 15
print(result)  # Output: 15
```

```
15
```

1. Lambda functions are generally less readable than named functions and should be used sparingly to avoid making the code harder to understand.

Less readable result = list(filter(lambda x: x % 2 == 0, nums)) # More readable def is_even(x): return x % 2 == 0 result = list(filter(is_even, nums))