LAMBDA AND SORTED WITH KEY_VHA

December 14, 2023

1 Benefits of using a Function

- Code Modularity
- Code Readibility
- Code Reusability

Lambda Function A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

2 syntax:

3 lambda a,b: a+b

4 lambda keyword : creates the lambda expression

5 parameters(a,b) : one or more parameters are supported. must be separated by a comma(,) and no parentheses

6 colon: This is a cue for the expression

7 expression(a+b): must be a single valid python expression

[1]:
```
# x -> x^2
lambda x:x**2
```

[1]:
```
<function __main__.<lambda>(x)>
```

[2]:
```
# x,y -> x+y
a = lambda x,y:x+y
a(5,2)
```

[2]:
```
7
```

8 Diff between lambda vs Normal Function

No name

lambda has no return value(infact,returns a function)

lambda is written in 1 line

not reusable

9 Then why use lambda functions?

They are used with HOF

[3]:
```
sub_string=lambda string: string in "welcome ljiet lju"
sub_string("lju")
```

[3]: True

[7]:
```
a=lambda x:'even' if x%2==0 else "odd"
print(a(4))
```

```
even
```

[8]:
```
add=lambda a,b:a+b
add(3,4)
```

[8]: 7

[9]:
```
(lambda a,b:a*b)(5,7)
```

[9]: 35

[10]:
```
product=lambda x,y,z:x+(y*z)
print(product(y=52, x=10, z=2))
print(product(4,2,3))
```

```
114
10
```

[11]:
```
add=lambda x,y=15,z=24:x+y+z
print(add(1))
print(add(1,2))
print(add(y=15,z=20,x=5))
print(add(2,5,7))
```

```
40
27
40
14
```

10 higher order function

A function is called Higher Order Function if it contains other functions as a parameter or returns a function as an output i.e, the functions that operate with another function are known as Higher order Functions. It is worth knowing that this higher order function is applicable for functions and methods as well that takes functions as a parameter or returns a function as a result. Python too supports the concepts of higher order functions.

Properties of higher-order functions:

A function is an instance of the Object type.

You can store the function in a variable.

You can pass the function as a parameter to another function.

You can return the function from a function.

You can store them in data structures such as hash tables, lists,

[12]:
```
# Example

def square(x):
  return x**2

def cube(x):
  return x**3

# HOF
def transform(f,L):
  output = []
  for i in L:
    output.append(f(i))

  print(output)

L = [1,2,3,4,5]
transform(square,L)
```

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

[13]:
```
L = [1,2,3,4,5]
transform(cube,L)
```

```
[1, 8, 27, 64, 125]
```

[14]:
```
transform(lambda x:x**3,L)
```

```
[1, 8, 27, 64, 125]
```

[18]:
```
# higher_ord_fun= lambda x,fun:x+fun(x)
def higher_ord_fun(x,y):
```

```
    y=y(3)
    return x+y
higher_ord_fun(20,lambda x:x*x)
```

[18]: 29

11 map

12 filter

13 reduce

[19]:
```
list1=[10,40,56,27,13,15,70]
divide=list(filter(lambda x: (x%4==0),list1))
print(divide)
```

```
[40, 56]
```

[20]:
```
# numbers greater than 5
L = [3,4,5,6,7]

list(filter(lambda x:x>5,L))
```

[20]: [6, 7]

[21]:
```
# fetch fruits starting with 'a'
fruits = ['apple','guava','cherry']

list(filter(lambda x:x.startswith('a'),fruits))
```

[21]: ['apple']

[22]:
```
list2=[10,40,56,27,13,15,70]
double=list(map(lambda x:x*2,list2))
print(double)
```

```
[20, 80, 112, 54, 26, 30, 140]
```

[23]:
```
list3=[2,5,10,6,4]
cube=list(map(lambda x:x**3,list3))
print(cube)
```

```
[8, 125, 1000, 216, 64]
```

[24]:
```
# odd/even labelling of list items
L = [1,2,3,4,5]
list(map(lambda x:'even' if x%2 == 0 else 'odd',L))
```

[24]:
```
['odd', 'even', 'odd', 'even', 'odd']
```

[25]:
```
from functools import reduce
list4=[2,5,10,6,4,2]
sum=reduce(lambda x,y:x*y,list4)
print(sum)
```

```
4800
```

[26]:
```
def quadratic(a,b,c):
    return lambda x: a*x**2+b*x+c
print(quadratic(2,3,5)(1))
```

```
10
```

[27]:
```
# fetch names from a list of dict

users = [
    {
        'name':'Rahul',
        'age':45,
        'gender':'male'
    },
    {
        'name':'Nitish',
        'age':33,
        'gender':'male'
    },
    {
        'name':'Ankita',
        'age':50,
        'gender':'female'
    }
]

list(map(lambda users:users['gender'],users))
```

[27]:
```
['male', 'male', 'female']
```

[28]:
```
# sum of all item
import functools

functools.reduce(lambda x,y:x+y,[1,2,3,4,5])
```

[28]:
```
15
```

[29]:
```
# find max
functools.reduce(lambda x,y:x if x>y else y,[23,11,45,10,1])
```

[29]: 45

14 sorted(itreable, key=none, reverse=False)

reverse=True/False

key=none

[30]:
```
l1=[1,2,3,4,5,6]
print(sorted(l1))
t1=(1,2,3,4,5,6)
print(sorted(t1))
d1={1:'a',2:'c',3:"b"}
print(sorted(d1.items()))
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[(1, 'a'), (2, 'c'), (3, 'b')]
```

[31]:
```
d1={1:'a',2:'c',3:"b"}
print(sorted(d1.items(),key=lambda x:x[1]))
print(d1.items())
```

```
[(1, 'a'), (3, 'b'), (2, 'c')]
dict_items([(1, 'a'), (2, 'c'), (3, 'b')])
```

[32]:
```
t=[(1,15),(2,11),(3,17),(4,12)]
def second(z):
    return z[1]
print(sorted(t,key=second))
```

```
[(2, 11), (4, 12), (1, 15), (3, 17)]
```

[33]:
```
t=[(1,15),(2,11),(3,17),(4,12)]
d=dict(t)
print(d)
print(sorted(d.items(),key=lambda x:x[1]))
print(d.items())
```

```
{1: 15, 2: 11, 3: 17, 4: 12}
[(2, 11), (4, 12), (1, 15), (3, 17)]
dict_items([(1, 15), (2, 11), (3, 17), (4, 12)])
```

[34]:
```
t=[(1,15),(2,11),(3,17),(4,12)]
print(sorted(t,key=lambda x:x[1]))
```

```
[(2, 11), (4, 12), (1, 15), (3, 17)]
```

[40]:
```
t=["vishal","vishal10","vishal9"]
print(sorted(t,key=len,reverse=True))
```

```
['vishal10', 'vishal9', 'vishal']
```

[41]:
```
t=["vishal","vishal10","vishal9"]
print(sorted(t,key=len))
```

```
['vishal', 'vishal9', 'vishal10']
```

[ ]:
```

```