Lambda Functions or Lambda Expressions

Lambda function is anonymous function; it is a function which does not any name.
Lambda functions are called higher order functions.
A function which is send as an argument to another function is called higher order function.

Syntax:
lambda arg1,arg2:expression

lambda expression can be defined,

1. With arguments
2. Without arguments

Lambda function without arguments not receives any values.

Example:

```
a=lambda:print("This is lambda Expression")
a()
a()
a()
```

Output:

```
======== RESTART: F:/python6pmaug/funtest51.py =======
This is lambda Expression
This is lambda Expression
This is lambda Expression
```

Lambda function with arguments receive values

Example:

```
a=lambda x,y:print(x,y)
a(10,20)
a(100,200)
```

Output:

10 20
100 200

Example:

```
def calculator(n1,n2,a):
   res=a(n1,n2)
   return res

def main():
    res1=calculator(10,5,lambda x,y:x+y)
    res2=calculator(10,4,lambda x,y:x-y)
    print(f'Result1 {res1}')
    print(f'Result2 {res2}')
main()
```

Output:

```
Result1 15
Result2 6
```

Example:

```
def filter_data(l,f):
    l1=[]
    for value in l:
        if f(value):
            l1.append(value)
    return l1


def main():
    list1=[12,15,18,9,5,7,3,12,14,16,20,22,24]
    list2=filter_data(list1,lambda num:num%2==0)
    list3=filter_data(list1,lambda num:num%2!=0)
    print(list1)
    print(list2)
    print(list3)
    list_str=['a','b','c','d','A','B','C','D']
    list_str1=filter_data(list_str,lambda s:s.islower())
    print(list_str)
    print(list_str1)
main()
```

Output:

```
[12, 15, 18, 9, 5, 7, 3, 12, 14, 16, 20, 22, 24]
```

[12, 18, 12, 14, 16, 20, 22, 24]
[15, 9, 5, 7, 3]
['a', 'b', 'c', 'd', 'A', 'B', 'C', 'D']
['a', 'b', 'c', 'd']

What is difference between a function and lambda function?

Function	**Lambda function**
Function is defined with “def” keyword	Lambda function is defined with “lambda” keyword
A function can defined with multiple statements	Lambda function is defined with only one statement
We can use return statement inside function to return value	We cannot use return statement inside lambda expression
A function cannot be defined as higher order function	Lambda function can be defined as higher order function

Function Recursion or Recursive functions

Calling function by itself is called recursive call or recursive functions.
In function recursion calling function and called function both are same.

Example:

```
def fun1():
    print("inside fun1")
    fun1() # recursive call

def main():
    fun1()
main()
```

Output:

```
inside fun1
inside fun1
inside fun1
inside fun1
inside fun1
inside fun1
inside fun1
inside fun1
inside fun1
```

```
inside fun1
inside fun1
inside fun1
inside fun1
inside fun1
Traceback (most recent call last):
  File "F:/python6pmaug/funtest55.py", line 7, in <module>
    main()
```

to work with recursion we required 3 statements

1. Initialization statement
2. Condition
3. Update statement

Function recursion is evaluated using a data structure stack. Stack follows LIFO (LAST IN FIRST OUT)

Example:

```
def print_num(num):
    if num<=3:
        print(num)
        print_num(num+1)
    print("nit")


def main():
    print_num(1)

main()
```

Output:

```
1
2
3
nit
nit
nit
nit
```

Example:

```
# create a recursive function to find factorial
# of input number


def factorial(n):
    if n==0:
        return 1
    else:
        return n*factorial(n-1)


def main():
    num=int(input("enter any number"))# 3
    res=factorial(num)
    print(f'factorial of {num} is {res}')

main()
```

Output:

```
======== RESTART: F:/python6pmaug/funtest57.py ========
enter any number4
factorial of 4 is 24

======== RESTART: F:/python6pmaug/funtest57.py ========
enter any number0
factorial of 0 is 1
```

Example:

```
c=0
def count_digits(num):
    if num>0:
        global c
        c+=1
        count_digits(num//10)


def main():
    count_digits(1234)
    print(f'count of digits {c}')
```

main()

Output:
count of digits 4

Monkey Patching
In Python, the term monkey patch refers to dynamic (or run-time) modifications of a class or module.

Monkey patching is a technique used to dynamically update the behavior of a piece of code at run-time. A monkey patch is a way to extend or modify the runtime code of dynamic languages without altering the original source code.