Batch: P5	**Roll No.: 16010422185**
Experiment / assignment / tutorial No. - 5	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Recursion and Lambda Function

AIM: To implement recursion function and lambda function

Expected OUTCOME of Experiment:

CO2: Use different Decision making statements and Functions in Python.

Resource Needed: Python IDE

Theory:

1. Python Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

1.1 Creating a Function

In Python a function is defined using the *def* keyword:

Example:
```
def my_function():
    print("Hello from a function")
```

1.2 Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

1.3 Parameters or Arguments?

The terms *parameter* and *argument* can be used for the same thing: information that is passed into a function. From a function's perspective:
A parameter is the variable listed inside the parentheses in the function definition.
An argument is the value that is sent to the function when it is called.

1.4 Number of Arguments

By default, a function must be called with the correct number of arguments i.e. if your function expects 2 arguments; you have to call the function with 2 arguments, not more, and not less.

1.5 Keyword Arguments

You can also send arguments with the key = value syntax.
This way the order of the arguments does not matter.

1.6 Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments will be passed into your function, add two asterisk: ** before the parameter name in the function definition.
This way the function will receive a dictionary of arguments, and can access the items accordingly

1.7 Default Parameter Value

The following example shows how to use a default parameter value.
If we call the function without argument, it uses the default value:

1.8 Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

1.9 Return Values

To let a function return a value, use the return statement:

1.10 The pass Statement

Function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

2. Recursion

Python also accepts function recursion, which means a defined function can call itself.
Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.
The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

To a new programmer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

3. 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.

Syntax of Lambda Function is given below

lambda arguments *:* expression

Lambda functions can take any number of arguments:

Problem Definition:

1. In below table input variable, python code and output column is given. You have to complete blank cell in every row.

S.No	Python Code	Output
1.	def my_function(fname, lname): print(fname + " " + lname) my_function("Amit", "Kumar")	Amit Kumar
2.	def my_function(fname, lname): print(fname + " " + lname) my_function("Emil")	TypeError Traceback (most recent call last) <ipython- input-9- 49655f8043b e> in <module> 2 print(fname + " " + lname) 3 ----> 4 my_function ("Emil") TypeError: my_function () missing 1 required positional argument: 'lname'

3.	def my_function(*kids): print("The youngest child is " + kids[2]) my_function("Emil", "Tobias", "Linus")	The youngest child is Linus
4.	def my_function(college3, college2, college1): print("The Best college is " + college3) my_function(**?**)	The Best college is KJSCE
5.	def my_function(**country= "Norway"**): print("I am from " + country) my_function("Sweden") my_function("India") my_function() my_function("Brazil")	I am from Sweden I am from India I am from Norway I am from Brazil

6.	def tri_recursion(k):
	  if(k > 0):
	  result = k + tri_recursion(k - 1)
	  print(result)
	
	  else:
	  result = 0
	
	  return result
	
	print("Recursion Example Results")
	tri_recursion(6)
		Recursion Example Results
		1
		3
		6
		10
		15
		21

7.	print((lambda x: x*2) (9))	18
8.	twice = lambda x: x*2 print(twice(9))	18

2. Write a Python program using a recursive function that takes a string as input from the user and displays whether the string is Palindrome or not.

3. Write a Python program to separate out even and odd numbers from the list entered by user by using Lambda function

Books/ Journals/ Websites referred:

1. Reema Thareja, *Python Programming: Using Problem Solving Approach*, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, *Python Programming: A modular Approach*, Pearson India, Second Edition 2018,India

Implementation details:

Code 1:

Untitled-1.py > ...

```
1   def pal(st):
2       if(len(st) <= 1):
3           return True
4       if st[0] == st[len(st)-1]:
5           return pal(st[1:len(st)-1])
6       else:
7           return False
8
9   st=input("Enter string : ")
10
11  if pal(st):
12      print("Entered string is Palindrome")
13  else:
14      print("Entered string is Not Palindrome")
15
16
```

Code 2:

```python
nums = [int(i) for i in input("Enter multiple values: ").split()]
#nums = [1,2,3,4,5,6,7,8,9,10]
print("Original list of integers: ")
print(nums)
print("\nEven numbers from the said list: ")
even_nums = list(filter(lambda x: x%2 == 0, nums))
print(even_nums)
print("\nOdd numbrers from the said list: ")
odd_nums = list(filter(lambda x: x%2 !=0, nums))
print(odd_nums)
```

Output(s):

Code 1:ss

```
PS C:\Users\Ant218\Desktop\Parth>  c:; cd 'c:\Users\Ant218\Desktop\Parth'; & 'C:\Users\Ant218\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users
\Ant218\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '49860' '--' 'c:\Users\Ant218\Desktop\P
arth\Untitled-1.py'
Enter string : 45654
Entered string is Palindrome
```

```
PS C:\Users\Ant218\Desktop\Parth>  c:; cd 'c:\Users\Ant218\Desktop\Parth'; & 'C:\Users\Ant218\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users
\Ant218\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '49869' '--' 'c:\Users\Ant218\Desktop\P
arth\Untitled-1.py'
Enter string : 256492
Entered string is Not Palindrome
```

Code 2:

```
PS C:\Users\Ant218\Desktop\Parth>  c:; cd 'c:\Users\Ant218\Desktop\Parth'; & 'C:\Users\Ant218\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users
\Ant218\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '49983' '--' 'c:\Users\Ant218\Desktop\P
arth\Untitled-1.py'
Enter multiple values: 1 2 3 4 5 6 7 8 9 10 11
Original list of integers:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Even numbers from the said list:
[2, 4, 6, 8, 10]

Odd numbrers from the said list:
[1, 3, 5, 7, 9, 11]
```

Conclusion: In this experiment we learned how to create a function and implement recursion function and lambda function in python. We also learned how to write the code of palindrome and how to separate even and odd numbers using lambda function in python.

Post Lab Descriptive Questions

1. Write a python program to calculate factorial using recursion
   Ans: Code :

```
C: > Users > User > exp 5 lq 1.1.py > ...
def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)
num = int(input("Enter a number: "))
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    print("The factorial of",num,"is",recur_factorial(num))
```

Output :

```
PS C:\Users\User>  c:; cd 'c:\Users\User'; & 'C:\Users\User\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\User\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '53309' '--' 'c:\Users\User\exp 5 lq 1.1.py'
Enter a number: 4
```

```
PS C:\Users\User>  c:; cd 'c:\Users\User'; & 'C:\Users\User\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\User\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '53330' '--' 'c:\Users\User\exp 5 lq 1.1.py'
Enter a number: -4
Sorry, factorial does not exist for negative numbers
```

2. What are the common functional programming methods that use lambdas?

Ans: In functional programming, lambda functions are commonly used as a way to create anonymous functions on-the-fly. Here are some of the common functional programming methods that use lambda functions:

Map: The map() function takes a lambda function and applies it to each element of an iterable. It returns a new iterable with the results.

Filter: The filter() function takes a lambda function and applies it to each element of an iterable. It returns a new iterable with only the elements that satisfy the condition in the lambda function.

Reduce: The reduce() function takes a lambda function and applies it to the first two elements of an iterable, then applies it to the result and the next element, and so on until all elements have been processed. It returns a single value.

Sort: The sort() method of a list takes a lambda function that specifies the key to use for sorting. It returns the sorted list.

Any and All: The any() and all() functions take a lambda function and apply it to each element of an iterable. any() returns True if any element satisfies the condition in the lambda function, while all() returns True if all elements satisfy the condition.

Date: **Signature of faculty in-charge**