Python FSD

User Defined Function

USER DEFINED FUNCTIONS

- A function is a block of organized code that is used to perform a single task. They provide better modularity for your application and reuse-ability.
- A function takes arguments and return the value.
- Syntax of the function

```
def functionName(parameterList):
    statement1;
    statement2;
```

SIMPLE USER DEFINED FUNCTIONS

```
def display():
    print("This is normal function ");
display();
```

FUNCTION WITH PASSING PARAMETER

- For function we can pass the zero or one or many parameter list

```
def say_hello(name):
    print("Welcome to user defined function "+name);
say_hello("Akash");
```

CONTINUE…

- Function with more than one parameter and display result in proper format.
- Passing the parameter with order as well as unorder with name of the parameter.

```
def emp_info(id,name,salary):
    print("Employee details");
    print("id is ",id,"name is ",name,"salary is ",salary);
    print(f'Id is {id} Name is {name} Salary is {salary}')
emp_info(100,"Ravi",24000);
emp_info(id=101,name="Ramesh",salary=26000);
emp_info(name="Lokesh",id=102,salary=28000);
```

FUNCTION WITH PASSING PARAMETER WITH RETURN VALUE

- We can write the function with passing parameter or no passing parameter and return the value of any types.

```
def sum_of_numbers(a,b):
    sum = a+b;
    return sum;
result = sum_of_numbers(10,20);
print("sum of two number is ",result);
```

PYTHON FUNCTION WITH ARBITRARY ARGUMENTS.

- Sometimes, we do not know in advance the number of arguments that we are going to pass to function. Python allows us to handle this kind of situation through functions calls with arbitrary number of arguments.

```
def stdInfo(id,name,*marks):
    print('Id is ',id)
    print('Name is ',name)
    for mark in marks:
        print('Mark ',mark)
stdInfo(1,"Raj",45)
stdInfo(2,'Seeta')
stdInfo(3,'Veeta',45,78,34)
stdInfo(4,'Rajesh',56,78,34,25)
```

PYTHON RECURSIVE FUNCTION

- We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive function.

```
def fact(num):
    if num==1:
        return 1;
    if num<=0:
        return 0;
    if num>1:
        return num*fact(num-1)
print('Fact of 0',fact(0))
print('Fact of 1',fact(1))
print('Fact of 2',fact(2))
print('Fact of 3',fact(3))
```

LAMBDA FUNCTION

- Lambda functions are similar to user-defined functions but without a name. They're commonly referred to as anonymous functions.
- Lambda functions are efficient whenever you want to create a function that will only contain simple expressions – that is, expressions that are usually a single line of a statement. They're also useful when you want to use the function once.
- Syntax
- lambda : arguments(s):expression
  - lambda is a keyword
  - arguments is placeholder, which is use to hold the value.
  - expression is the code you want to execute in the lambda funtions.

SIMPLE LAMBDA EXPRESSION

```
def sayHello():
    print("This is normal function");
sayHello();
```

```
sayHello=lambda : print("lambda expression");
sayHello();
```

LAMBDA EXPRESSION WITH PASSING PARAMETER

```
def addNumber(a,b):
    return a+b;
print(addNumber(10,20));
```

```
addNumber2= lambda a,b:a+b;
print(addNumber2(100,200));
```

LAMBDA EXPRESSION WITH IMMEDIATE INVOKE FUNCTION STYLE

- Some time if we want to execute the function only one time. that time we can use the IIFE function ie Immediate invoke function expression

```
(lambda x : x * 2)(3)
```

WHEN TO USE LAMBDA FUNCTION

- You should use the lambda function to create simple expressions. For example, expressions that do not include complex structures such as if-else, for-loops, and so on.
- So, for example, if you want to create a function with a for-loop, you should use a user-defined function.

COMMON USE CASES FOR LAMBDA FUNCTIONS

- Filter(): When you want to focus on specific values in an iterable, you can use the filter function. The following is the syntax of a filter function:
  - filter(function,iterable);
    - list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    - filter(lambda x: x % 2 == 0, list1)
    - list(filter(lambda x: x % 2 == 0, list1))
- map(): You use the map() function whenever you want to modify every value in an iterable.
  - map(function,iterable);
    - list1 = [2, 3, 4, 5]
    - list(map(lambda x: pow(x, 2), list1))