FUNCTIONS & ARRAYS –

- **Definitions -** We write functions that return values, which we will call fruitful functions. We have seen the return statement before, but in a fruitful function the return statement includes a return value.
- This statement means: ***"Return immediately from this function and use the following expression as a return value."***
- Any function that returns a value is called **Fruitful function**.
- A Function that does not return a value is called a **void function**.
- **Return values -** The Keyword return is used to return back the value to the called function.

```
# returns the area of a circle with the given radius
def area(radius):
        temp = 3.14 * radius**2
return temp
print(area(4))
```

NOTE: Sometimes it is useful to have multiple return statements, one in each branch of a conditional statement.

```
def absolute_value(x):
        if x < 0:
                return -x
        else:
                return x
```

NOTE: Since these return statements are in an alternative conditional, only one will be executed.

NOTE: As soon as a return statement executes, the function terminates without executing any subsequent statements. Code that appears after a return statement, or any other place the flow of execution can never reach, is called **dead code.**

NOTE: In a fruitful function, it is a good idea to ensure that every possible path through the program hits a return statement.

```
def absolute_value(x):
        if x < 0:
                return -x
        if x > 0:
                return x
```

NOTE: **This function is incorrect because if x happens to be 0, both conditions is true, and the function ends without hitting a return statement. If the flow of execution gets to the end of a function, the return value is None, which is not the absolute value of 0.**

 # Write a Python function that takes two lists and returns True if they have at least one common member.

```
def common_data(list1, list2):
    for x in list1:
        for y in list2:
            if x == y:
                result = True
                return result
print(common_data([1,2,3,4,5], [1,2,3,4,5]))
print(common_data([1,2,3,4,5], [1,7,8,9,510]))
print(common_data([1,2,3,4,5], [6,7,8,9,10]))
```

- **Parameters:** Parameters are passed during the definition of function while Arguments are passed during the function call.

```
#here a and b are parameters
def add(a,b):
    result=add(12,13)
    print(result)
```

 # To display vandemataram by using function use no args no return type

```
#function defination
def display():
    print("vandemataram")
    print("i am in main")

#function call
display()
print("i am in main")
```

Types of Functions –

#Type1 : Without return type, without parameters

```
def Fun1() :
    print("function 1")
Fun1()
```

#Type 2: Without return type, with parameters

```
def fun2(a) :
        print(a)
fun2("hello")
```

#Type 3: With return type, without parameters

```
def fun3():
        return "welcome to python"
print(fun3())
```

#Type 4: With return type, with parameters

```
def fun4(a):
        return a
print(fun4("python is better then c"))
```

Local and Global Scope –

- **Local Scope:** A variable which is defined inside a function is local to that function. It is accessible from the point at which it is defined until the end of the function, and exists for as long as the function is executing.

- **Global Scope:** A variable which is defined in the main body of a file is called a global variable. It will be visible throughout the file, and also inside any file which imports that file. The variable defined inside a function can also be made global by using the global statement.

 # Create a global variable

```
x = "global"
def f():
        print("x inside :", x)
f()
print("x outside:", x)
```

 # Create a Local variable

```
def f1():
        y = "local"
        print(y)
f1()
```

```
# use Global variable and Local variable with same name
x = 5
def f4():
        x = 10
        print("local x:", x)

f4()
print("global x:", x)
```

Function Composition –

Having two (or more) functions where the output of one function is the input for another. So for example if you have two functions FunctionA and FunctionB you compose them by doing the following.

FunctionB(FunctionA(x))

Example 1:

```
#create a function compose2
>>> def compose2(f, g):
        return lambda x:f(g(x))

>>> def d(x):
        return x*2

>>> def e(x):
        return x+1

>>> a=compose2(d,e)          # FunctionC = compose(FunctionB,FunctionA)
>>> a(5)                     # FunctionC(x) 12
```

Recursion - Recursion is the process of defining something in terms of itself.

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 functions.

- **NOTE:** Factorial of a number is the product of all the integers from 1 to that number.
- **NOTE:** For example, the factorial of 6 (denoted as 6!) is 1*2*3*4*5*6 = 720.

```
# Write a program to factorial using recursion
def fact(x):
        if x==0:
                result = 1
        else :
                result = x * fact(x-1)
        return result
print("zero factorial",fact(0))
print("five factorial",fact(5))
```