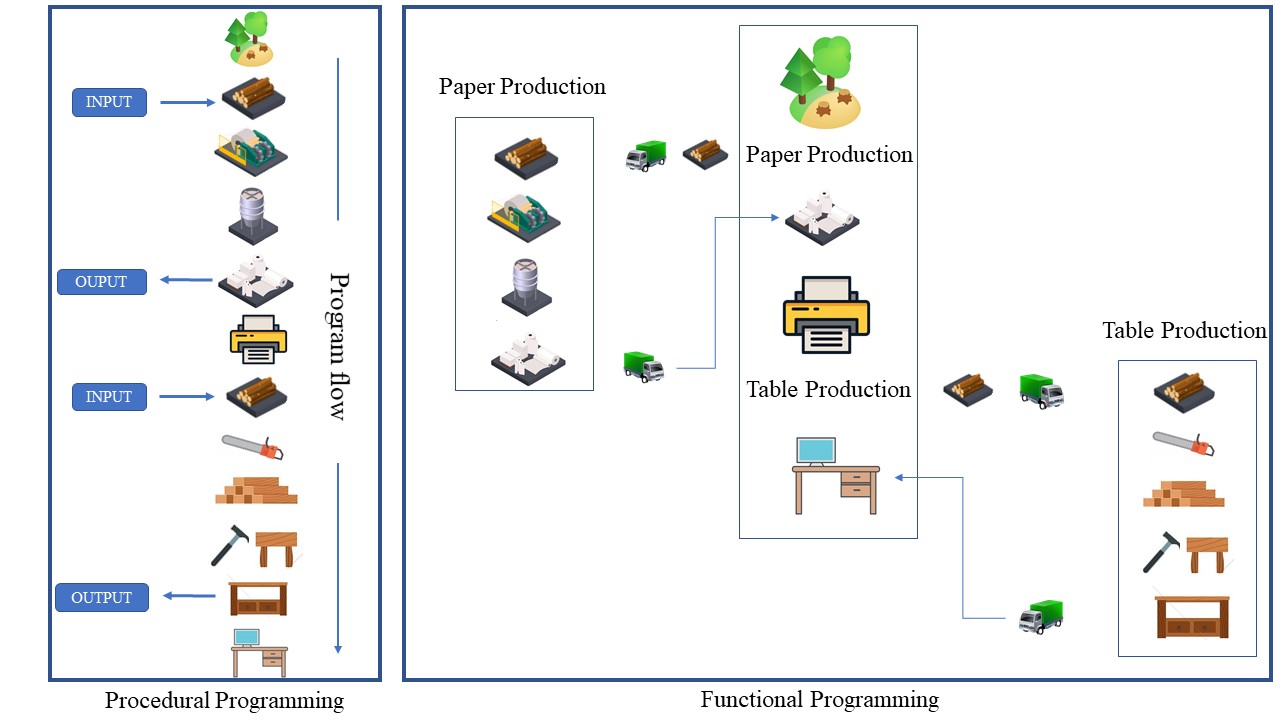
# Session 6 - Functions

1. Introduction to Functions
2. Function definition
3. Passing in arguments
4. Return keyword
5. Scope

**All the programs that we wrote till now can be called procedural programs. As our code was being executed in a single flow. But when we have a big task to complete the code starts getting bigger which causes the code to be less readable.**

**To Tackle this issue Functional programming was introduced.**

## Procedural Vs. Functional Programming



In the above image, the left-hand side block represents a procedural programming approach. And we can make out that procedural programming for big or multiple tasks can become very complicated. So we go for an alternate approach the functional programming.

In functional programming, we can split the code into 2 parts

* main program
* functions

Each function has a **unique name** which is used by the main program to **call** the function so that the task defined inside that function can be executed. In the above image, we have split the procedural code into 2 functions and a main program. This makes the code look a lot cleaner. The main program calls the function name and gives the raw input so that the function can process the input and return some output.

*The Input and Output functionalities are optional.*

Another advantage of using function is code reuse. If we refer to the above image of functional programming since we have defined a function called paper production every time the main program wants paper it can call the function name. If the same thing was to be achieved using procedural programming we would have to copy and paste the entire paper production tasks. This return increases the code size.

## Function definition

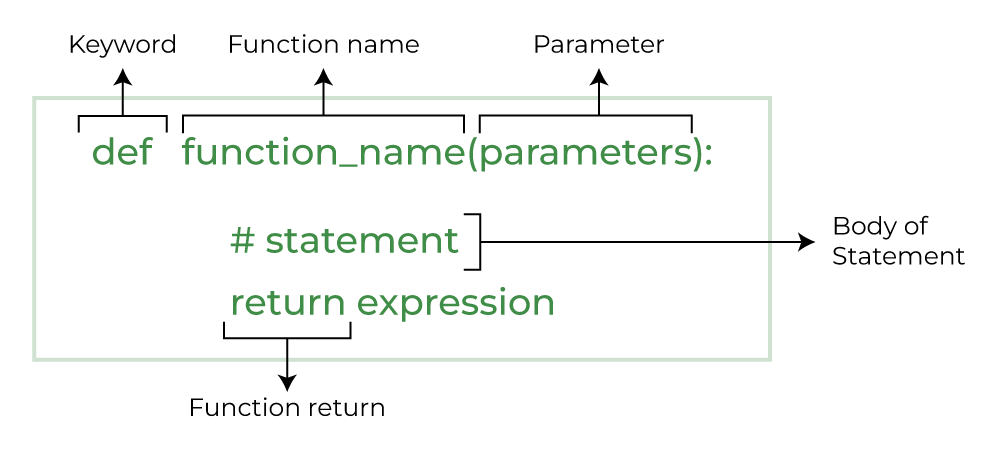
**Function Syntax**

**def** name\_of\_function(arg1,arg2):  
  
 *# Do stuff here*  
 *# Do stuff here*  
 *# Do stuff here*  
   
 **return** statement *#Return desired result*

Don't get carried away by the complexity of the syntax. As we discussed earlier input and output from functions are not mandatory. The arg1 and arg2 are the inputs and the return keyword is used as an output from the function.

So let's start with a simpler version of functions

**Function syntax**

****

**def** function\_name():  
 *# do something*  
 *# do something*

### A simple print 'hello world' function

**def** say\_hello():  
 print('hello')  
 print('world')

We will get no output when we run the above cell. All that we are doing is function definition.

The def in the function syntax stands for definition. we are supposed to call the function to run the lines of code in the function. **Calling a function simply means typing the name of the function followed by open and close brackets.**

say\_hello()

hello  
world

## Arguments or parameters

**NOTE: Arguments and parameters are the same thing** More often than not we will be required to give input to the function so that the function can act on the input and give us a desired output.

Arguments are nothing but variables defined in the function definition. We assign values to these variables when we call the function.

**Syntax**

**def** function\_name(arg1):  
 *# do something with arg1*  
 print(result)

**def** add\_10(x):  
 s = x+10  
 print(s)

The above function takes in a value for the argument x adds 10 to the x and prints it.

add\_10(100)

110

The Entire idea behind using the function with arguments is to be able to call the function with different inputs.

add\_10(1000)

1010

### If the function definition argues it is mandatory to give a value to that argument while calling the function we will get an error

look at the below example

add\_10()

---------------------------------------------------------------------------  
TypeError Traceback (most recent call last)  
<ipython-input-6-199c52168f5e> in <module>  
----> 1 add\_10()  
  
TypeError: add\_10() missing 1 required positional argument: 'x'

We can have multiple arguments in the brackets

**def** my\_add(a,b,c):  
 print(a+b+c)

my\_add(10,100,1000)

1110

### The number of values in the bracket should be equal to the number of argument variables in the function definition

## Using return

Let's see some examples that use a return statement. return allows a function to *return* a result that can then be stored as a variable, or used in whatever manner a user wants.

**def** add\_10(x,y):  
 z = x+y  
 **return** z

a = add\_10(4,5)  
print(a)

9

The obvious question in your mind is why would we use a return statement when we could have printed the output in the function itself.

Yes, We could have done that but what if we are using the function just to do some operation and we want to use the value of the calculated output outside of the function

**def** add\_10(x,y):  
 z = x+y  
 print(z)

add\_10(100,10)  
  
print(z+1)

110

---------------------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-15-9b14b4bd2a62> in <module>  
 1 add\_10(100,10)  
 2   
----> 3 print(z+1)  
  
NameError: name 'z' is not defined

The above error brings us to the next big topic

## Scope in python

# Scope: A variable is only available from inside the region it is created. This is called scope.

There are two types of scopes. ***1. Local scope: A variable created inside a function belongs to the local scope of that function, and can only be used inside that function. 2. Global scope: A variable created in the main body of the Python code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local.***

To keep things simple all we need to know is that a variable that is defined inside a function only exists inside the function. So for the function to be able to get data out we have to use return statements and store the resulting output into a new variable when calling the function.

# REVISION

**1. Function definition 2. Passing in arguments 3. return keyword 4. scope**



### TASK 1

#### Define a function to determine if a number is prime or even

**def** is\_prime(num):  
 *'''*  
 *Naive method of checking for primes.*   
 *'''*  
 **for** n **in** range(2, num):  
 **if** num % n == 0:  
 print(num,' is not prime')  
 **break**  
 **else**: *# If never mod zero, then prime*  
 print(num,' is prime!')

### TASK 2

### 2. Create a function to convert "Fahrenheit" to "Celcius"

def fahrenheit\_to\_celsius(Fahrenheit):

celsius = (Fahrenheit - 32) \* 5/9

return celsius

fahrenheit\_temperature = int(input('enter the temperature in Fahrenheit))

celsius\_temperature = fahrenheit\_to\_celsius(fahrenheit\_temperature)

print(fahrenheit\_temperature, "degrees Fahrenheit is equal to",celsius\_temperature)

### 

### 

### 

****

# HOMEWORK

**1. Write a function that accepts a single argument and check if it is a Palindrome**

* The function should return True or False depending on the result

Eg of a palindrome: MALAYALAM

**2. Write a function that accepts a single argument and prints the factorial of the passed number**

**3. Write a function that prints the Fibonacci series up to the passed argument number**

*#TASK 1:*  
  
**def** isPalindrome(s):  
 new\_s = s[::-1]  
 **return** new\_s  
  
s = "malayalam"  
ans = isPalindrome(s)  
   
**if** ans == s:  
 print("Yes")  
**else**:  
 print("No")

Yes

*#TASK 2:*  
  
**def** fact(n):  
 fact = 1  
 **for** i **in** range(1,n+1):  
 fact = fact \* i  
   
 print(fact)  
   
n = int(input("enter your number "))  
fact(n)

enter your number 5  
120

*#TASK 3:*  
  
**def** Fibonacci(n):  
 **if** n < 0:  
 print("Incorrect input")  
   
 **elif** n == 0:  
 **return** 0  
   
 **elif** n == 1 **or** n == 2:  
 **return** 1  
   
 **else**:  
 **return** Fibonacci(n-1) + Fibonacci(n-2)  
  
print(Fibonacci(9))

34

# TASK 4:

Write a Python function to find the Max of three numbers

# TASK 5:

Write a Python function to check whether a number falls in a given range

# TASK 6:

Write a Python program to print even numbers from a given list

# TASK 7:

Write a Python function that takes a list and returns a new list with unique elements of the first list.

# TASK 8:

Write a program to create function calculation() such that it can accept two variables and calculate addition and subtraction. Also, it must return both addition and subtraction in a single return call.

# TASK 9:

write a code to find the function to check whether x is even or odd

# TASK 10:

Write a Python function that accepts a string and calculates the number of upper-case letters and lower-case letters

# TASK 11:

# Project -- Tic-Tac-Toe Game

**Come up with the steps that will help us create a Tic-Tac-Toe game using the things we have learned till now.**