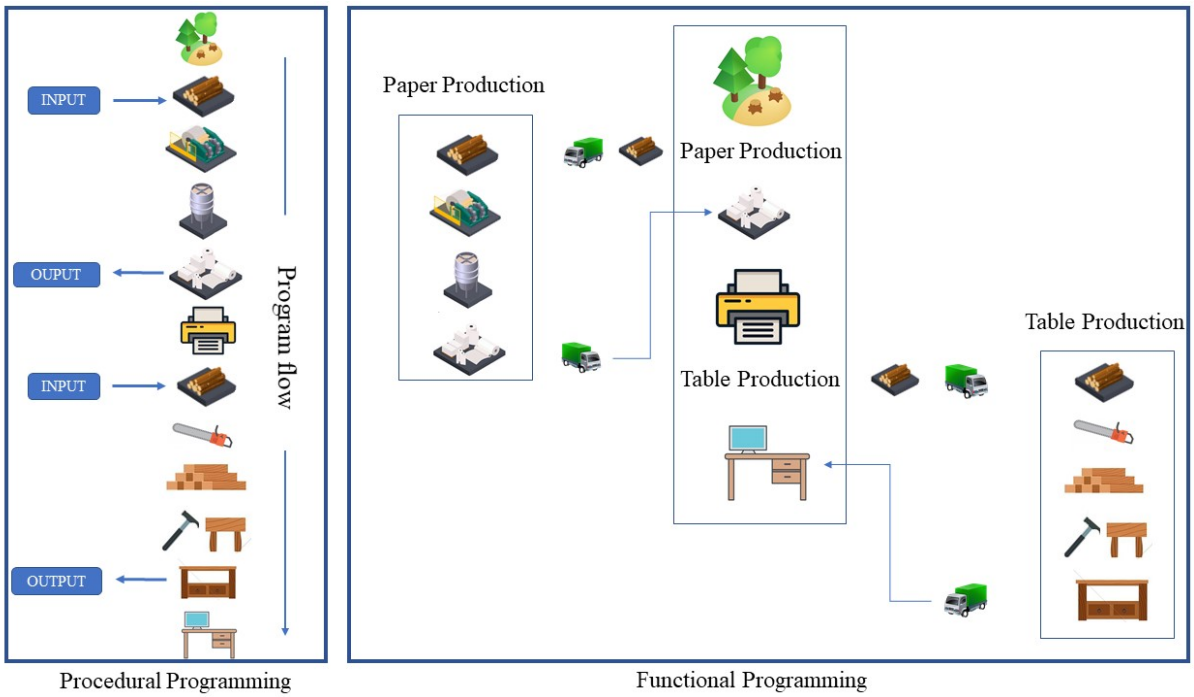
Session 6 - Functions

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 big task to complete the code starts getting bigger this causes the code to be less readable.

To Tackle this issue Fucntional 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 progamming for big or multiple tasks can become very complicated. So we go for an alternate approcah the functional programming.

In functional programming we can split the code into 2 parts

- main program
- fucntions

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

The Input and Output functionalities are optional.

Another advantage of using fucntion is code reuse. If refered to the above image of fucntional programming since we have deined a fucntion called `paper prodcuction` every time the main program wants paper it can call the fucntion name. If the same thing was to be acheived using procedural programmming we would have to copy and paste the entire paper production tasks. This inturns 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 ouput 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 lets start with a simpler version of functions

Function syntax

```
def function_name():
    # do something
    # do something
```

A simple print 'hello world' function

In [1]:
```
def say_hello():
    print('hello')
    print('world')
```

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

The def in the function syntax stands for definition.
we are supossed 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.

In [2]:
```
say_hello()
```

```
hello
world
```

Arguments or parameters

NOTE : Arguments and parameters are the same thing
More often than not we will be requiring to give input to the function so that the funciton 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)
```

In [3]:
```
def add_10(x):
    s = x+10
    print(s)
```

The above function take in a value for the arguments `x` and adds 10 to the `x` and prints it.

In [4]:
```
add_10(100)
```

```
110
```

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

In [5]:
```
add_10(1000)
```

```
1010
```

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

look at the below example

In [6]:
```
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

In [7]:
```
def my_add(a,b,c):
    print(a+b+c)
```

In [8]:
```
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 example 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.

In [12]:
```
def add_10(x,y):
    z = x+y
    return z
```

In [13]:
```
a = add_10(4,5)
print(a)
```

```
9
```

The obvious question in your mind is why would we use return statement when we cound 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 opeartion and we want to use value of the calculated output outside of the function.

In [14]:
```
def add_10(x,y):
    z = x+y
    print(z)
```

In [15]:
```
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

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

TASK 1

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

In [11]:
```
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!')
```

HOMEWORK

1.Write a fucntion which accepts single argument and check if its a Palindrome

- the fucntion should return True or False depending on the result

Eg of palindrome : MALAYALAM

2.Write a fucntion which accepts single argument prints the factorial of the passed number

3.Write a fucntion prints fibonacci series upto the passed argument number

In [18]:
```
#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
```

In [19]:

```
#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
```

In [20]:

```
#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
```

Project -- Tic-Tac-Toe Game

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

In [ ]:

In [ ]: