Default Parameter in function:

Arbitrary Arguments, *args:
If we do not know how many arguments that will be passed into a function, add a * before the parameter name in the function definition.This way the function will receive a *tuple* of arguments, and can access the items accordingly.

Syntax

def Function_Name (p1,p2,p3,...):

Statements

function call

Function_Name(a1,a2,a3,...)

a = arguments (value) ,p = parameters (variables)

```
def Func(*args):
   print(args)
   size = len(args)
   print("Received",size,"elements")

Func(1,3,9,7)
Func(1,3)
```

Default value concept.

```
def Func(n1= 0, n2 = 0, n3 = 0):
  print(n1,n2,n3)

Func() #by default arguments prints 0 0 0
Func(3,5,7) # print 3 = n1 , 5 = n2 ,7 = n3 (override the default values)
Func(1,3)   # override the current values with n1 = 1 , n2 = 3
```

Non- default values(parameters) can be used with default values in a sequence first non-default value then default values.

```
def Func(n1, n2 = 0, n3 = 0):
  print(n1,n2,n3)

Func(3,5,7) # print 3 = n1 , 5 = n2 ,7 = n3 (override the default values)
Func(1,3)
```

Keyword Arguments: we can also send arguments with the *key* = *value* syntax.This way the order of the arguments does not matter

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

The pass statement can be used for in loop, Functions, decision making and conditional statement.

Example

```
for i in range(10):
  pass
```

Recursion: A defined function call itself. This has the benefit of meaning that we can loop through data to reach a result. Recursion based on two conditions.

https://github.com/ibraheem-02/Python_Programs/blob/main/Recursion.ipynb

1.Base situation/condition/case.(end of recursion).

2.Recursive case (repeat itself) Recursive case end as loop or vise versa.

1.Base case

Let's find the factorial of a number take number from user for example user enter a number 5.

Algorithm

```
 fact(5) = 5*fact(5-1)
        fact(4) = 4*fact(4-1)
                  fact(3) = 3*fact(3-1)
                            fact(2) = 2*fact(2-1)
                                      fact(1) = 1*fact(1-1)
                                                fact(0) = 1
```

Python Program

```
def fact(n):
    if n==0:
        return 1
    return n*fact(n-1)
n = int(input("Enter a Number: "))
result = fact(n)
print("Factorial of ",n,"is",result)
```

File Handling: File handling refers to the process of performing operations on a file such as creating, opening, reading, writing and closing it, through a programming interface. It involves managing the data flow between the program and the file system on the storage device, ensuring that data is handled safely and efficiently. Python has several functions for creating, reading, updating, and deleting files.

The key function for working with files in Python is the open ( ) function.

The open ( ) function takes two parameters; *filename*, and *mode.*

Syntax
Obj_Name = open(File_Name, mode)

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition we can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Opening a File in Python

To open a file we can use open() function, which requires file path and mode as arguments:

Example.

```
# read the file from same folder
F1 = open("Class_Work/File.txt", "r")
print(F1.read())
F1.close()
```

Reading a File

Reading a file can be achieved by **file.read()** which reads the entire content of the file. After reading the file we can close the file using file.close() which closes the file after reading it, which is necessary to free up system resources.

```
# read a file character by character
F2 = open("Class_Work/File.txt", "r")
print(F2.read(10)) #read the starting 10 characters including spaces
# read a file line by line
F2.close()
F3 = open("Class_Work/File.txt", "r")
print(F3.readline()) #read the  line
print(F3.readline()) #read the second line
F3.close()
```

Writing to a File

Writing to a file is done using **file.write()** which writes the specified string to the file. If the file exists, its content is erased. If it doesn't exist, a new file is created.

Example: Writing to a File in Write Mode (w)

```
# writing a data on a file
F4 = open("Class_Work/File.txt", "a") #write mode
F4.write("\nThis is the new line added to the file")
F4.close()
```

Writing to a File in Append Mode (a)

It is done using file.write() which adds the specified string to the end of the file without erasing its existing content.

Example: For this example, we will use the Python file created in the previous example.

```
# writing a data on file
F1 = open("Class_Work/File.txt", "a") #append mode
F1.write("Muhammad Khalid")
F1.close()
```

Closing a File

Closing a file is essential to ensure that all resources used by the file are properly released. file.close() method closes the file and ensures that any changes made to the file are saved