Working with files in Python

Working with files in Python involves reading from or writing to files on your computer. You can open files using the open() function, read their contents, modify data, or create new files. After working with a file, it's important to close it using the close() method to free up system resources.

```
In [ ]: # reading a file and read the data
        x = open(r"C:\Users\EURO TEC COMPUTERS\OneDrive\Documents\Python Scripts","r")
        data = x.read()
        print(data)
        x.close()
```

```
In [6]: file_path = "my_file.txt"  # Specify the file name and path

        # Open the file in write mode (this will create the file if it doesn't exist)
        with open(file_path, "w") as file:
            file.write("Hello, this is some content written to the file"
                       "my pharmacy in bwp located"
                      "how are you")

        print(f"File '{file_path}' has been created and written to.")
```

```
File 'my_file.txt' has been created and written to.
```

```
In [7]: # Reading a line in the file
        x = open(r"my_file.txt","r")
        data = x.readline()
        print(data)
        x.close()
```

```
Hello, this is some content written to the filemy pharmacy in bwp locatedhow are you
```

In [9]:
```
# Reading all the lines
x = open(r"my_file.txt","r")
data = x.readlines()
print(data)
x.close()
```

```
['Hello, this is some content written to the filemy pharmacy in bwp locatedhow are you']
```

In [12]:
```
# Reading files piece by piece
x = open(r"my_file.txt","r")
for line in x:
        print(line)
x.close()
```

```
Hello, this is some content written to the filemy pharmacy in bwp locatedhow are you
```

In [13]:
```
# Writing Files in Python
x = open(r"my_file.txt","w")
x.write("This is another line")
x.close()
```

2- Importing code with python

In Python, the import statement is used to bring functionality from external modules or libraries into your code. It allows you to use functions, classes, and variables defined in those modules within your own Python program.

In [17]:
```
# For example, if you want to use the math module to perform mathematical operations,
# you would use the import statement like this:

import math

result = math.sqrt(25)
print(result)
```

```
5.0
```

In [18]:
```
# Using from to import
# Another way to import a function is to use from
from math import sqrt
sqrt(16)
```

Out[18]:
```
4.0
```

3_ Functions in Python

In Python, a function is a reusable block of code that performs a specific task. It allows you to encapsulate functionality and call it whenever needed.

In [21]:
```
# example

def function_name(parameters):
    """
    Docstring: Description of the function.
    """
    # Function code here
    # ...
    return result  # Optional return statement

# Example function
def greet(name):
    """Prints a greeting message."""
    print(f"Hello, {name}!")

# Calling the function
greet("Alice")
```

```
Hello, Alice!
```

In [23]:
```
def a_function():
    print("we just created a function!")
```

In [24]:
```
# Empty function
def empty_function():
    pass
```

In [26]:
```
# add arguments to functions
def add(a,b):
    return a + b
add(1,2)
```

Out[26]:
```
3
```

In [27]:
```
add(a=2, b=3)
total = add(b=4, a=5)
print(total)
```

```
9
```

Keyword Arguments

Keyword arguments in Python allow you to pass arguments to a function using the parameter names, which makes the code more readable and allows you to specify arguments in any order.

In [28]:
```
# Function with keyword arguments
def print_person_info(name, age, city):
    print(f"Name: {name}, Age: {age}, City: {city}")

# Using keyword arguments
print_person_info(name="Alice", age=30, city="New York")
```

```
Name: Alice, Age: 30, City: New York
```

In [29]:
```
# example
def keyword_function(a=1 , b=2):
    return a+b
keyword_function(b=4, a=5)
```

Out[29]:
```
9
```

```
In [31]: # Default argument keyword!
         def keyword_function(a=1 , b=2):
             return a+b
         keyword_function()

Out[31]: 3
```

```
In [34]: # Lets create a function which has both regular and keyword arguments
         def mixed_function(a, b=2, c=3):
             return a+b+c
```

```
In [37]: mixed_function(1, b=4, c=5)

Out[37]: 10
```

```
In [40]: # Examples
         def many(*args, **kwargs):
             print(args)
             print(kwargs)
         many(1,2,3, name="Ali" , job="Cyber Warrior", cast = "rana")

         (1, 2, 3)
         {'name': 'Ali', 'job': 'Cyber Warrior', 'cast': 'rana'}
```

Scope*

In Python we have a concept of scope just like other programming languages. Scope tells us when a variable is available to use and where. We have to define the variable inside a function, as those variables can only be used inside that function only. Once that function ends, that variable cannot be used again and is known as out of scope.

In [44]:
```
# Examples of scope and global
def function_a():
    global a
    a = 1
    b = 2
    return a+b
def function_b():
    c = 3
    return a+c
print(function_a())
print(function_b())
```

```
3
4
```

In [ ]: