Task-2{Basmala}

<<<Functions>>>

The diff. Between print() and return:

- **Print**: provides output to the console.
- Return: provides the value you can store, work with, and code later.

Variable scope:

- *"Scope is essential to understanding how information is passed throughout programs"*
- Reusing names for objects is OK as long as you keep them in a separate scope.
- Python doesn't allow functions to modify variables that are outside the function's scope, so a better way would be to pass the variable as an argument and reassign it outside the function.
- But functions can still print the content of a global variable.

Documentation:

- Functions are especially readable because they often use documentation strings, or docstrings.
- Docstrings are a type of comment used to explain the purpose of a function, and how it should be used.
- surrounded by triple quotes (''')

Lambda Expressions:

- Used to create anonymous functions.
- Anonymous functions. functions that don't have a name, they are helpful for creating quick functions that aren't needed later in your code, and also useful for higher order functions, or functions that take in other functions as arguments.

- lambda expressions aren't ideal for complex functions but can be very useful for short, simple functions.

`Map():-`

- is a higher-order function that works **as an iterator to return a result after applying a function to every item of an iterable as: (**tuple, lists, etc.).
- It is used when you want to apply a single transformation function to all the iterable elements.

`Filters():`

- a higher-order built-in function that takes a function and iterable as inputs
- **filters the given sequence with the help of a function that tests each element in the sequence to be true or not.**

<<Scripting>>

`The input():-`

- A function that takes in whatever the user types and stores it as a string (by default).

"We can also interpret user input as a Python expression using the built-in function `eval`"

`eval():-`

- This function evaluates a string as a line of Python.

Errors & Exceptions:

- **Syntax errors:** occur when Python can't interpret our code, since we didn't follow the correct syntax for Python.
- **Exceptions:** occur when unexpected things happen during the execution of a program, even if the code is syntactically correct.

- ValueError: this is **an exception that occurs when a function receives an argument of the correct data type but an inappropriate value**.
- AssertionError**:** assert is a simple statement with the following syntax: assert expression[, assertion_message] Here, expression can be any valid Python expression or object, which is then tested for truthiness.
- **If an expression is false, then the statement throws an AssertionError**
- KeyError: is **an exception that occurs when an attempt is made to access an item in a dictionary that does not exist.**
- TypeError is **an exception that occurs when the data type of an object in an operation is inappropriate**
- IndexError: **when an item from a list is attempted to be accessed that is outside the index range of the list**.

Try Statement:

  - `try`: This is the only mandatory clause in a `try` statement. The code in this block is the first thing that Python runs in a `try` statement.
  - `except`: If Python runs into an exception while running the `try` block, it will jump to the `except` block that handles that exception.

  - `else`: If Python runs into no exceptions while running the `try` block, it will run the code in this block after running the `try` block.
  - `finally`: Before Python leaves this `try` statement, it will run the code in this `finally` block under any conditions, even if it's ending the program.
  - you can still access error messages, even if you handle them to keep your program from crashing!

Reading and writing files:

Reading a File:

- 1. First open the file using the built-in function, `open`. This requires a string that shows the path to the file. The `open` function returns a file object, which is a Python object through which Python interacts with the file itself.

- Assign this object to any variable.
- There are optional parameters you can specify in the `open` function. One is the mode in which we open the file.
- To read we use `r` or read only.
- `r` is actually the default value for the mode argument.
- `read` method is used to access the contents from the file object.
- This `read` method takes the text contained in a file and puts it into a string.
- 1. When finished with the file, use the `close` method to free up any system resources taken up by the file.

Writing to a File:

- 1. Open the file in writing `('w')` mode. If the file does not exist, Python will create it for you.
- If you open an existing file in writing mode, any content that it had contained previously will be deleted.
- If you're interested in adding to an existing file, without deleting its content, you should use the `append ('a')` mode instead of write.

With:

“auto-closes a file for you once you're finished using it.”

- the indented code is executed, in this case, reading from the file. Now, we don’t have to call f.close()!

Importing files:

- We can import Python code from other scripts
- If the Python script you want to import is in the same directory as your current script, you just type `import` followed by the name of the file, without the .py extension.
- Modules are just Python files that contain definitions and statements.

main block:

- To avoid running executable statements in a script when it's imported as a module in another script, include these lines in an `if __name__ =="__main__"` block.
- Or include them in a function called main() and call this in the `if main` block.
- built-in `__name__` variable is just set to the name of that module.
- the condition `if __name__ == "__main__"` is just checking whether this module is the main program.

Importing:

- `from` `module_name` `import` `object_name` to import an individual function or class from a module
- `from` `module_name` `import` `first_object, second_object` To import multiple individual objects from a module
- `import` `module_name` `as` `new_name` To rename module
- `from` `module_name` `import` `object_name` `as` `new_name` To import an object from a module and rename it
- A sub-module is specified with the usual dot notation.

Experminting with an interpreter:

- To quit the Python interactive interpreter, use the command `exit()` or hit `ctrl-D` on mac .