**Day 11 Assignment - 15/12/2023 - Vamsi Viswanadham**

Worked on the Below Concepts

* **Mapping function**

1. Python’s map() is a built-in function used to process and transform all the items in an iterable without using an explicit for loop, a technique commonly known as mapping.
2. map() is useful when you need to apply a transformation function to each item in an iterable and transform them into a new iterable. map() is one of the tools that support a functional programming style in Python.

Eg:

def addition(n):

return n + n

# We double all numbers using map()

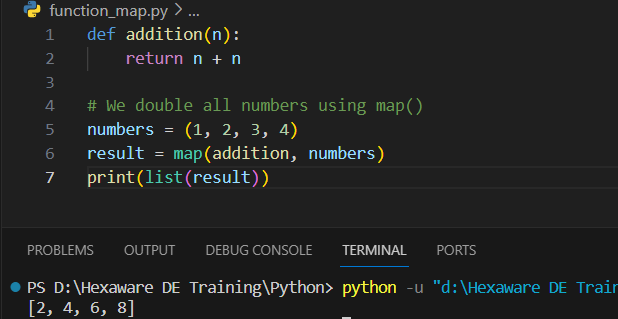
numbers = (1, 2, 3, 4)

result = map(addition, numbers)

print(list(result))

Here we are creating a new function to add two numbers, and trying to use the map() function to map this functions to the numbers tuple and get the result.

Output:



* **String Function**

capitalize(): Capitalizes the first letter of the string.

upper(): Converts all characters to uppercase.

lower(): Converts all characters to lowercase.

strip(): Removes leading and trailing whitespaces.

split(): Splits the string into a list based on a delimiter.

replace(): Replaces a substring in the string with another substring.

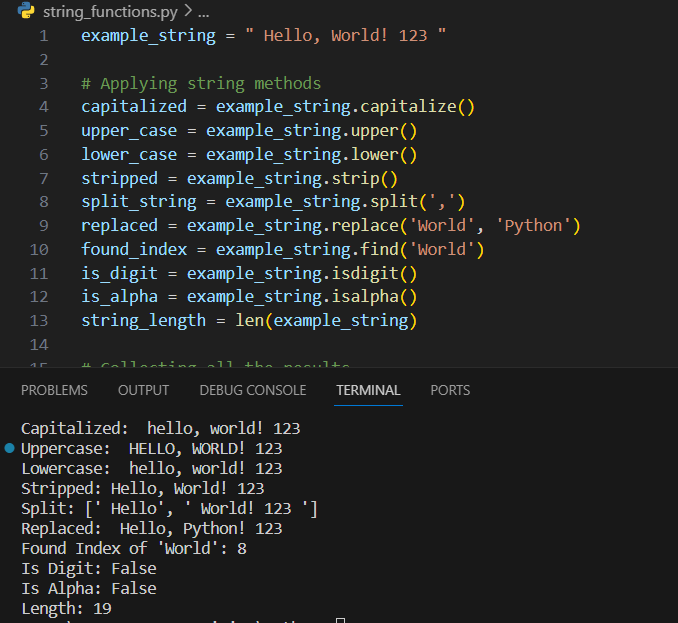
find(): Returns the lowest index of the substring if found.

isdigit(): Returns True if all characters in the string are digits.

isalpha(): Returns True if all characters are alphabetic.

len(): Returns the length of the string.

Here is an example for explaining string functions:

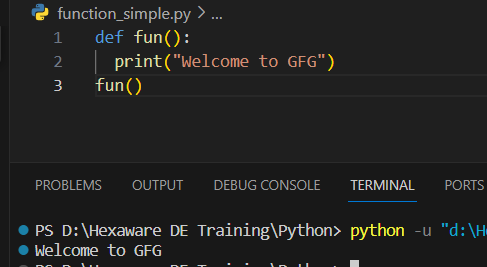


* **Python Functions**

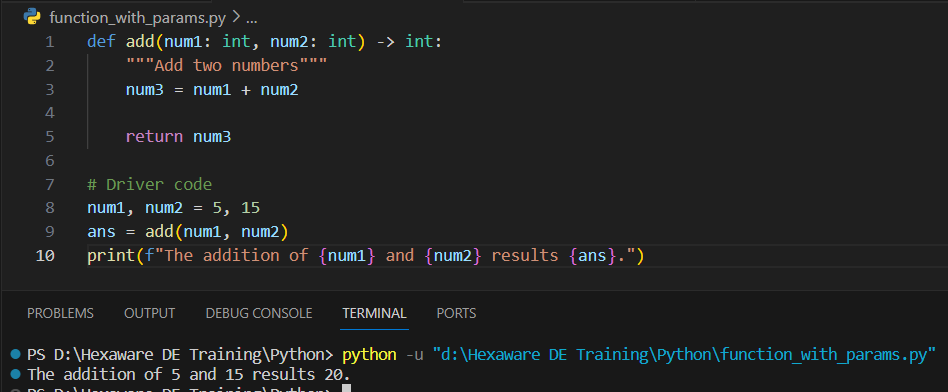
Function is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Eg:

A simple function starts with a ‘def’ keyword and then function name followed by parameters(optional) and a colon(:).



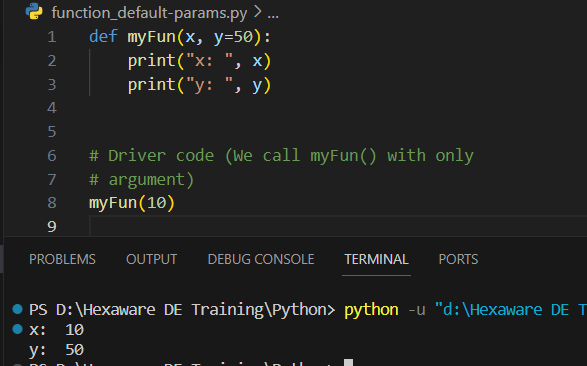
Functions with parameters:



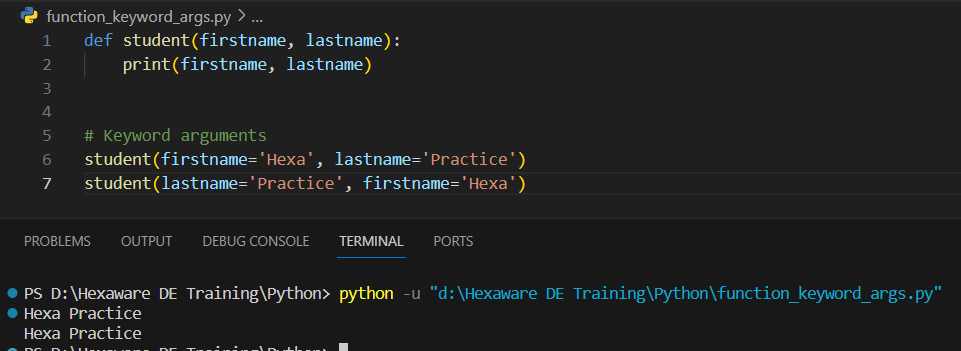
Here is a simple example for function with parameters where we have pass the parameters separated with comma(,) after the function name in the function definition, then inorder to pass these parameters, you can pass them in the function calling statement as shown in the above figure.

* **Default Argument Values**

A function can have a default argument that is, if the argument value isnt provided in the function calling time, then the default provided value will be considered as the value for that parameter. Eg: Here we have taken ‘y’ as a default parameter.



* **Keyword Arguments**

****

The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

* **Special parameters**
  + Positional-Only Parameters (/): In Python 3.8 and later, you can specify that

some parameters are positional-only. If positional-only, the parameters' order matters and the parameters cannot be passed by keyword.

Eg:

def concat(a, b, /, separator):

return a + separator + b

# Correct usage

print(concat("Hello", "World", separator=" "))

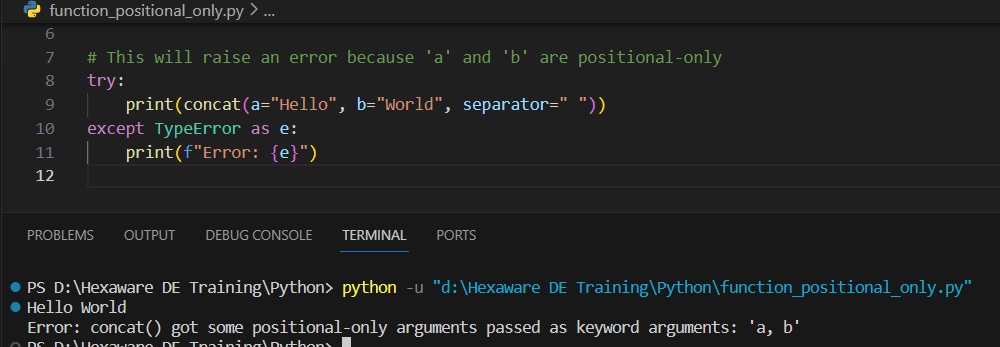
# This will raise an error because 'a' and 'b' are positional-only

try:

print(concat(a="Hello", b="World", separator=" "))

except TypeError as e:

print(f"Error: {e}")



In this function, a and b are positional-only arguments. You can't use them as keyword arguments. The separator argument is not positional-only, so it can be used either as a keyword or positional argument.

* + Keyword-Only Arguments (\*): After the \*, all the following parameters are keyword-only, meaning they can only be passed by keyword, not positionally.

Eg:

def concat(a, b, /, separator):

return a + separator + b

# Correct usage

print(concat("Hello", "World", separator=" "))

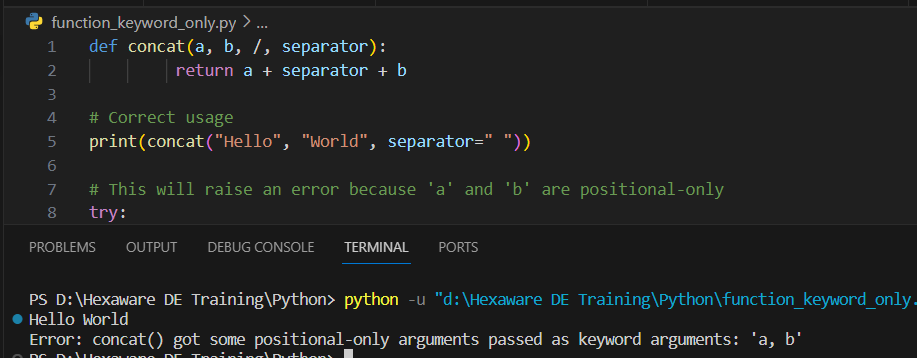
# This will raise an error because 'a' and 'b' are positional-only

try:

print(concat(a="Hello", b="World", separator=" "))

except TypeError as e:

print(f"Error: {e}")

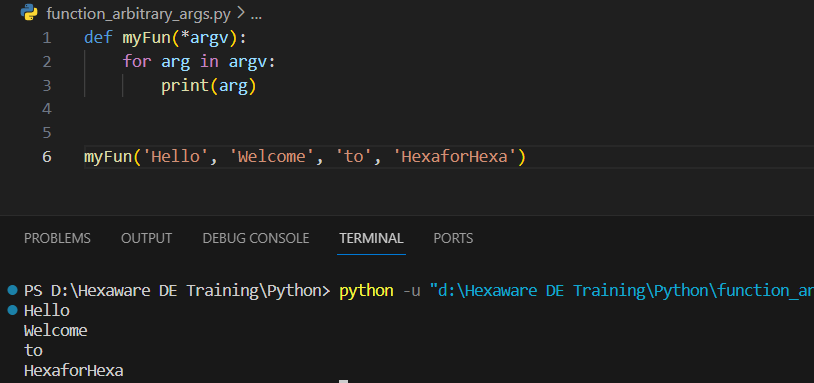


In this function, a and b are positional-only arguments. You can't use them as keyword arguments. The separator argument is not positional-only, so it can be used either as a keyword or positional argument.

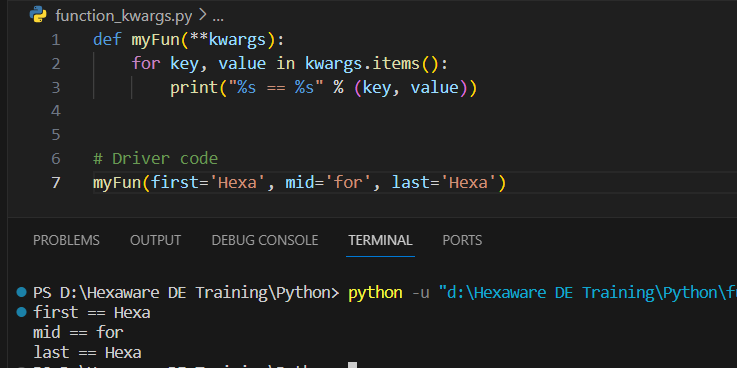
* **Arbitrary Argument Lists**

In Python Arbitrary Keyword Arguments, \*args, and \*\*kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:

\*args in Python (Non-Keyword Arguments):



\*\*kwargs in Python (Keyword Arguments):



* **Lambda Expressions**

In Python, an anonymous function means that a function is without a name. As we already know the def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.

Here is an example of the lambda function: Here we have just created a lambda function that takes x as an input and multiply itself three times and store that in the cube\_v2 variable.

