Week 3 - Python Functions, Scope and Errors

In week 3 exercise, we learnt about create and use functions and variable scope as well as error fixing in python.

Functions:

Python functions are reusable blocks of code that perform a specific task. They are defined with the def keyword, can take inputs (arguments), and may return outputs.

Scope:

Python scope determines the visibility and lifespan of variables within different parts of the code. Variables can exist in local, enclosing, global, or built-in scopes, impacting where they can be accessed or modified.

Error:

Python errors are issues that arise when code encounters invalid operations, leading to exceptions or syntax errors.

# Example code for Functions:

#Greet Friends

def greet\_user(first\_name , last\_name , university = "UWS" ):

print(f"Hello {first\_name} {last\_name} from  {university}")

greet\_user("Aman" , "Misra" , "UWS London")

Hello Aman Misra from UWS London

def greet\_friends(friends):

for name in friends:

print(f"Hello {name}")

friends\_list = ["Aman" , "Sreeraj" , "Prajwal"]

greet\_friends(friends\_list)

Hello Aman

Hello Sreeraj

Hello Prajwal

The **greet\_user** function is a simple yet effective way to generate customizable greeting messages. Its use of default parameters and string formatting makes it versatile and user-friendly.

**def greet\_user(first\_name, last\_name, university="UWS"):** Declares a function with three parameters, where university has a default value of "UWS".

def add(num1 , num2):

return num1 + num2

def multiply(num1 , num2):

return num1 \* num2

print(add(3 , 5))

print(multiply(1 , 5))

Here , add: A function to perform addition of two numbers ,

multiply: A function to perform multiplication of two numbers.

Both functions are designed to return the results of their operations. These results can be used immediately or stored for later use.

In Python, the **return** keyword is used within a function to send a value back to the calling context. Here’s how the **return** keyword is used in the given code.

# Tax Calculation

def calculate\_tax(income , tax\_rate):

tax\_amount = income \* tax\_rate

return tax\_amount

total\_money = calculate\_tax(50000 , 0.2)

print(total\_money)

In this code ,the function **calculate\_tax** takes two parameters: **income** and **tax\_rate**.

**tax\_rate**: Represents the tax rate as a decimal (e.g., 20% is 0.2)

**income**: Represents the amount of money on which tax is calculated.

The result of the calculation is sent back to the calling context using the **return** statement.

The returned value **(tax\_amount)** is stored in the variable **total\_money**.

The **calculate\_tax** program demonstrates effective use of functions, return values, and arithmetic operations.

Variable Scope:

def new\_function():

my\_new\_variable = 5

new\_function() # call the function. No problems here.

print(my\_new\_variable) # this will cause an error

Here,the provided Python code defines a function **new\_function** that declares a variable **my\_new\_variable** and assigns it a value of 5. After calling the function, there is an attempt to print the value of **my\_new\_variable.**

Variables in Python have a specific scope (the context in which they are accessible)

Variables defined inside a function (local variables) are accessible only within the function.

Function Execution:

When the function **new\_function** is called, the variable **my\_new\_variable** is created in the local scope of the function and is discarded once the function exits.

Error Explanation:

The attempt to access **my\_new\_variable** outside the function results in a **NameError**, as the variable is not defined in the global scope.

my\_new\_variable = 0

def new\_function():

my\_new\_variable = 5

new\_function()

print(my\_new\_variable)

The provided Python code defines a variable **my\_new\_variable** in the global scope, assigns it an initial value of 0, and then defines a function **new\_function** that declares a new variable with the same name **(my\_new\_variable)** and assigns it a value of 5. The function is called, and the program attempts to print the value of **my\_new\_variable**.

**Code Execution Analysis**

1. **my\_new\_variable = 0:**
   * A **global variable** named my\_new\_variable is initialized with the value 0.
2. **def new\_function(): my\_new\_variable = 5:**
   * Inside the function new\_function, a **local variable** **my\_new\_variable** is created and assigned the value 5.
   * This variable is local to the function and does not affect the global **my\_new\_variable**.
3. **new\_function():**
   * When the function is called, the local **my\_new\_variable** is assigned 5 but is discarded as soon as the function ends.
4. **print(my\_new\_variable):**
   * The global variable **my\_new\_variable** remains unchanged and retains its original value of 0.

Optional: Assertions and Errors:

# Error fixing

# pritn("Hello, World!") # error in pritn

print("Hello, World!")

# number1 = "5"

number1 = 5

number2 = 3

result = number1 + number2

print(f"The sum is {result}")

fruits = ["apple", "banana", "cherry"]

# print(fruits[3]) -> index out of bounds

print(fruits[2])

time = 11

if time < 12:

#print("Good morning!")  # no indentation/tab space

print("Good morning!")

Code Overview

The provided code initially contains several errors, both syntactical and logical. These errors have been fixed to ensure the code runs without issues. Here's the corrected version.

**Explanation of the Fixes**

1. **Error in pritn("Hello, World!")**:
   * Issue: The function name was misspelled as pritn instead of print.
   * Fix: Changed pritn to print.
2. **Type Mismatch in Addition**:
   * Issue: number1 was a string ("5"), and adding it to an integer (number2) caused a TypeError.
   * Fix: Changed number1 to an integer (5) to ensure compatibility during addition.
3. **Index Out of Bounds in print(fruits[3])**:
   * Issue: The fruits list has only three elements (index 0, 1, and 2), but the code attempted to access index 3, leading to an IndexError.
   * Fix: Changed the index to 2, which is valid and retrieves the last element of the list.
4. **Indentation Error in if Statement**:
   * Issue: The print statement under the if condition was not indented correctly, causing a SyntaxError.
   * Fix: Indented the print("Good morning!") statement properly under the if block.

In this code, the **print()** function is used to display the message **"Hello, World!"** on the screen.

Two variables, **number1** and **number2**, are assigned the values **5** and **3** respectively.

These numbers are added together, and the result **(8)** is stored in the variable result.

The **print()** function is used again to display the message "The sum is 8", where **{result}** is replaced by the value stored in result **(which is 8).**

The fruits list contains three items: **"apple",** **"banana",** and "**cherry".**

The **print()** function is used to display the third item in the list **(fruits[2]),** which is **"cherry".**

The variable time is set to **11.**

An if statement checks if time is less than **12**. Since **11** is less than **12**, it prints **"Good morning!".**

Summary:

Week 3 introduced essential programming concepts like functions, variable scope, and error handling. These skills form the foundation for writing clean, organized, and reliable Python code, setting the stage for tackling more advanced challenges.