Lab # 3: Learning Functions and Classes in Python

3.1 Objectives

- Develop and use reusable code by encapsulating tasks in functions.
- Package functions into flexible and extensible classes.

3.2 Pre-Lab

The code from the previous lab experiments was limited to short snippets. Solving more complex problems means more complex code - stretching over hundreds, to thousands, of lines, and—if you want to reuse that code—it is not convenient to copy and paste it multiple times. Worse, any error is magnified, and any changes become tedious to manage.

In Python, "functions" are complete suite of functions grouped around a set of related tasks is called a library or module. Libraries permit you to inherit a wide variety of powerful software solutions developed and maintained by other people.

Functions in Python are defined using the def keyword, and they play a crucial role in organizing code, promoting reusability, and improving readability. Here are key aspects and concepts related to functions in Python:

- **Function Definition:**
  Functions are defined using the "def" keyword, followed by the function name and a pair of parentheses. Any parameters the function takes are listed inside the parentheses.

```
def greetings(name):
    print(f"Hello, {name}!")

greetings("Alice")
```

- **Function Call:**
  To execute a function, you "call" it by using its name followed by parentheses. If the function takes parameters, you provide the values inside the parentheses.

```
def add(a, b):
    return a + b

result1 = add(3, 4)
```

- **Return Statement:**
  Functions can return a value using the **return** keyword. The returned value can be assigned to a variable or used directly.

```
def multiply(x, y):
    return x * y

result2 = multiply(5, 6)
```

- **Parameters and Arguments:**
  Parameters are variables listed in a function's definition. Arguments are the values passed to the function when it is called.

- **Default Parameters:**
  You can provide default values for parameters. If a value is not passed for a parameter, the default value is used.

```
def power(base, exponent=2):
    return base ** exponent

result4 = power(3)  # Uses default exponent of 2
```

- **Variable-Length Arguments:**
  Functions can accept a variable number of arguments using ***args** and ****kwargs**.

```
def sum_all(*numbers):
    return sum(numbers)

result5 = sum_all(1, 2, 3, 4)
```

- **Lambda Functions:**
  Also known as anonymous functions, these are small, one-line functions defined using the **lambda** keyword.

```
square = lambda x: x ** 2
result6 = square(5)
```

- **Scope:**

  Variables defined inside a function have local scope, meaning they are only accessible within that function. Variables defined outside functions have global scope.

```python
global_var = 10

def my_function():
    local_var = 5
    print(global_var)  # Accessible
    print(local_var)   # Accessible
```

3.3 In-Lab

In-Lab Task 1

Write a Python function called "count_even_numbers" that takes a list of integers as input and returns the count of even numbers in the list. Additionally, ensure that the function handles the case where the input list is empty. Follow the following instructions:

- **Define a function named "count_even_numbers" that takes one parameter: number_list (a list of integers).**
- **Inside the function, use a loop to iterate through each number in the list.**
- **Use a conditional statement to check if each number is even.**
- **If a number is even, increment a counter variable.**
- **After the loop, return the counter variable.**

```python
# ================================================================================
# In - Lab Task 1
# ================================================================================
# Function definition
def count_even_numbers(number_list):
    # Initialize counter variable
    even_count = 0

    # Loop through each number in the list
    for num in number_list:
        # Check if the number is even
        if num % 2 == 0:
            even_count += 1

    # Return the count of even numbers
    return even_count

# Test cases
numbers1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numbers2 = [11, 13, 15, 17]
numbers3 = []  # Empty list

# Display results
print(count_even_numbers(numbers1))  # Expected output: 5
print(count_even_numbers(numbers2))  # Expected output: 0
print(count_even_numbers(numbers3))  # Expected output: 0
```

In-Lab Task 2

Write a Python function calculate_gpa that takes a list of dictionaries, where each dictionary represents the marks of a student in 5 different courses (out of 100). Each dictionary should have keys 'name' and 'marks' (a list of 5 marks)

* Percentage Obtained in a Semester System	Grade	Grade Points
85 and above	A	4.00
80 84	A-	3.66
75 79	B+	3.33
71 - 74	B	3.00
68 70	B-	2.66
64 67	C+	2.33
61 63	C	2.00
58 60	C-	1.66
54 - 57	D+	1.30
50 53	D	1.00
Below 50	F	0.00

Calculate the Grade Point Average (GPA) for each student by averaging their Grade Points.

The function should return a list of dictionaries where each dictionary has keys 'name', 'grades' (a list of course grades), 'grade_points', and 'gpa'.


Create a dictionary which have at least record to five students (Name and Scores in 5 Courses)

3.3.1 Class

In Python, a class is a blueprint for creating objects. Objects are instances of a class, and each object can have attributes (characteristics) and methods (functions) associated with it. Classes provide a way to bundle data and functionality together. Here's a basic explanation of how classes work in Python:

- **Class Declaration: To create a class, you use the class keyword. Here's a simple example of a class:**

```
class Car:
    pass  # The 'pass' keyword is a placeholder for the class body
```

- **Attributes: You can define attributes (characteristics) inside the class. These attributes represent the data associated with objects of the class.**

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model
```

In this example, the **__init__** method is a special method called a constructor. It is called when an object is created. **self** refers to the instance of the class (the object), and you can set attributes like **make** and **model** for each object.

- **Methods: You can also define methods inside the class. Methods are functions associated with the class.**

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"{self.make} {self.model}")
```

The **display_info** method can be called on an instance of the **Car** class to display information about the car.

- **Creating Objects (Instances): Once a class is defined, you can create objects (instances) of that class.**

```
my_car = Car("Toyota", "Camry")
```

This creates an instance of the **Car** class called **my_car** with the specified make and model.

- **Accessing Attributes and Calling Methods: You can access attributes and call methods using the dot notation.**

```
my_car = Car("Toyota", "Camry")


print(my_car.make)  # Output: Toyota
my_car.display_info()  # Output: Toyota Camry
```

Here, **my_car.make** accesses the **make** attribute, and **my_car.display_info()** calls the **display_info** method.

In-Lab Task 3

Create a Python class named **student** to manage student information. The class should have the following attributes:

1. **name**: Name of the student.
2. **roll_number**: Roll number of the student.
3. **marks**: A list to store the marks of the student in different subjects.

The class should have the following methods:

1. **__init__**: A constructor to initialize the attributes.
2. **add_marks**: A method to add marks to the **marks** list.
3. **calculate_average**: A method to calculate and return the average marks of the student.

Create an instance of the **student** class, add some marks, and calculate the average.

3.4 Post-Lab

Post-Lab Task

Create a simple library management system using Python. Define a class **Book** to represent a book with attributes such as title, author, and availability status. Implement functions to borrow and return books.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____