

Assignment-19.4

Task 1:

Write a Python function print_numbers() that prints the first 10 natural numbers using a loop.

Code and Output:

```
def print_numbers():
    """Prints the first 10 natural numbers using a loop."""
    for i in range(1, 11):
        print(i)

print_numbers()
```

```
1
2
3
4
5
6
7
8
9
10
```

Translate the function into JavaScript as a reusable function printNumbers().

Javascript:

The screenshot shows a terminal window with two panes. The top pane is labeled "main.js" and contains the following code:

```
1 function printNumbers() {  
2     // Prints the first 10 natural numbers using a Loop.  
3     for (let i = 1; i <= 10; i++) {  
4         console.log(i); // prints i each time the loop runs  
5     }  
6 }  
7  
8 printNumbers(); // calls the function  
9
```

The bottom pane is labeled "input" and shows the terminal output:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Explanation:

K explanation

← Certainly! The code in cell 4e6a948d is a Python function called `print_numbers()`.

Here's a breakdown:

- `def print_numbers():` : This line defines a function named `print_numbers`. The code inside this function will only run when the function is called.
- `"""Prints the first 10 natural numbers using a loop."""` : This is a docstring, which explains what the function does.
- `for i in range(1, 11):` : This is a `for` loop.
 - `range(1, 11)` creates a sequence of numbers starting from 1 and going up to (but not including) 11. So, the sequence is 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.
 - The loop will iterate through each number in this sequence, assigning the current number to the variable `i` in each iteration.
- `print(i)` : Inside the loop, this line prints the current value of `i` to the console.
- `print_numbers()` : This line calls the `print_numbers` function, which executes the code defined within it.

So, when the cell is run, the `print_numbers()` function is called, and the loop iterates from 1 to 10, printing each number on a new line.



Task 2:

Write a Java method `checkNumber(int num)` that checks if a number is positive, negative, or zero.

Java:

The screenshot shows a Java development environment with the following details:

- Toolbar:** Run, Debug, Stop, Share, Save, Beautify.
- Code Editor:** The file `NumberChecker.java` contains the following code:

```
1 public class NumberChecker {
2
3     public static void checkNumber(int num) {
4         if (num > 0) {
5             System.out.println(num + " is a positive number.");
6         } else if (num < 0) {
7             System.out.println(num + " is a negative number.");
8         } else {
9             System.out.println(num + " is zero.");
10        }
11    }
12
13    public static void main(String[] args) {
14        checkNumber(10);
15        checkNumber(-5);
16        checkNumber(0);
17    }
18 }
```
- Output Console:** The output shows the results of the program execution:

```
10 is a positive number.
-5 is a negative number.
0 is zero.

...Program finished with exit code 0
Press ENTER to exit console.
```

Translate the method into a Python function `check_number(num)`.

Python:

```
❶ def check_number(num):
    """Checks if a number is positive, negative, or zero."""
    if num > 0:
        print(f"{num} is a positive number.")
    elif num < 0:
        print(f"{num} is a negative number.")
    else:
        print(f"{num} is zero.")

# Example usage:
check_number(10)
check_number(-5)
check_number(0)
```

→ 10 is a positive number.
-5 is a negative number.
0 is zero.

+ Code

+ Text

Call the function/method with different inputs and compare outputs.

```
❷ def check_number(num):
    """Checks if a number is positive, negative, or zero."""
    if num > 0:
        print(f"{num} is a positive number.")
    elif num < 0:
        print(f"{num} is a negative number.")
    else:
        print(f"{num} is zero.")

# Example usage with various inputs:
check_number(10)
check_number(-5)
check_number(0)
check_number(100)
check_number(-99)
check_number(0.5)
check_number(-0.1)

→ 10 is a positive number.
-5 is a negative number.
0 is zero.
100 is a positive number.
-99 is a negative number.
0.5 is a positive number.
-0.1 is a negative number.
```

Explanation:

❸ Explanation

→ Certainly! The code in cell a813674c defines a Python function called `check_number(num)` and then demonstrates how to use it with various inputs.

Here's a breakdown:

- `def check_number(num):` : This line defines a function named `check_number` that takes one argument, `num`.
- `"""Checks if a number is positive, negative, or zero."""` : This is a docstring explaining the function's purpose.
- `if num > 0:` : This is the first condition. It checks if the number `num` is greater than 0. If it is, the code inside this `if` block is executed.
 - `print(f'{num} is a positive number.')` : This line prints a message indicating that the number is positive, using an f-string to include the value of `num` in the output.
- `elif num < 0:` : This is an "else if" condition. If the first `if` condition (`num > 0`) is false, this condition is checked. It checks if `num` is less than 0. If it is, the code inside this `elif` block is executed.
 - `print(f'{num} is a negative number.')` : This line prints a message indicating that the number is negative.
- `else:` : This is the final condition. If both the `if` and `elif` conditions are false (meaning `num` is neither greater than nor less than 0), the code inside this `else` block is executed.
 - `print(f'{num} is zero.')` : This line prints a message indicating that the number is zero.

The lines after the function definition are examples of how to call the `check_number` function with different values (10, -5, 0, 100, -99, 0.5, and -0.1). Each call passes a different number to the `num` parameter of the function, and the function then prints the corresponding output based on whether the number is positive, negative, or zero.

Task 3:

Write a Python function factorial(n) that calculates factorial of a number using recursion

Python:

```
▶ def factorial(n):
    """
    Calculates the factorial of a non-negative integer using recursion.

    Args:
        n: A non-negative integer.

    Returns:
        The factorial of n.
    """

    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

    # Example usage:
print(factorial(5))
print(factorial(0))
```

→ 120
1

Translate the same into a C++ function int factorial(int n).

C++:

```
main.cpp
1 #include <iostream>
2
3 int factorial(int n) {
4     if (n < 0) {
5         // Factorial is not defined for negative numbers
6         return -1; // Or throw an exception
7     }
8     if (n == 0) {
9         return 1;
10    } else {
11        return n * factorial(n - 1);
12    }
13}
14
15 int main() {
16     std::cout << "Factorial of 5 is: " << factorial(5) << std::endl;
17     std::cout << "Factorial of 0 is: " << factorial(0) << std::endl;
18     std::cout << "Factorial of -1 is: " << factorial(-1) << std::endl; // Example with negative input
19     return 0;
20 }
```

```
Factorial of 5 is: 120
Factorial of 0 is: 1
Factorial of -1 is: -1

...Program finished with exit code 0
Press ENTER to exit console.
```

Explanation:

K explanation

◆ Certainly! The code in cell 2c6a6df5 defines a Python function called `factorial(n)` that calculates the factorial of a non-negative integer using recursion.

Here's a breakdown:

- `def factorial(n):` : This line defines a function named `factorial` that takes one argument, `n`.
- `"""Calculates the factorial of a non-negative integer using recursion."""` : This is a docstring explaining the function's purpose.
- `if n == 0:` : This is the base case for the recursion. The factorial of 0 is defined as 1. When `n` is 0, the function stops calling itself and returns 1.
- `return 1`: If `n` is 0, the function returns the value 1.
- `else:` : If the base case is not met (i.e., `n` is not 0), the code inside the `else` block is executed.
- `return n * factorial(n-1)` : This is the recursive step. The factorial of `n` is calculated as `n` multiplied by the factorial of `n-1`. The function calls itself with `n-1` as the argument. This continues until the base case (`n == 0`) is reached.

The lines after the function definition are examples of how to call the `factorial` function with inputs 5 and 0, and the `print()` function displays the returned results.

In essence, the recursion works like this for `factorial(5)`: `factorial(5)` calls `factorial(4)` | `factorial(4)` calls `factorial(3)` | `factorial(3)` calls `factorial(2)` | `factorial(2)` calls `factorial(1)` | `factorial(1)` calls `factorial(0)` | `factorial(0)` returns 1 (base case) | `factorial(1)` receives 1 and returns $1 \times 1 = 1$ | `factorial(2)` receives 1 and returns $2 \times 1 = 2$ | `factorial(3)` receives 2 and returns $3 \times 2 = 6$ | `factorial(4)` receives 6 and returns $4 \times 6 = 24$ | `factorial(5)` receives 24 and returns $5 \times 24 = 120$

And for `factorial(0)`: `factorial(0)` reaches the base case and returns 1 immediately.

Task 4:

Write a JavaScript function `printStudents(students)` that takes an array of student names and prints each name.

Javascript:

```
main.js
1 function printStudents(students) {
2   /**
3    * Prints each student name from an array.
4    * @param {string[]} students - An array of student names.
5   */
6   for (let i = 0; i < students.length; i++) {
7     console.log(students[i]);
8   }
9 }
10
11 // Example usage:
12 const studentNames = ["Alice", "Bob", "Charlie"];
13 printStudents(studentNames);

Alice
Bob
Charlie

...Program finished with exit code 0
Press ENTER to exit console.
```

Translate it into a Python function `print_students(students)` using a list.

Python:

```
❶ def print_students(students):
    """
    Prints each student name from a list.

    Args:
        students: A list of student names.
    """
    for student in students:
        print(student)

# Example usage:
student_names = ["Alice", "Bob", "Charlie"]
print_students(student_names)
```

→ Alice
Bob
Charlie

Explanation:

K Explanation

◆ Certainly! The code in cell 7140c818 defines a Python function called `print_students(students)` that iterates through a list of student names and prints each one.

Here's a breakdown:

- `def print_students(students):`: This line defines a function named `print_students` that takes one argument, `students`. This argument is expected to be a list.
- `"""Prints each student name from a list."""`: This is a docstring explaining the function's purpose.
- `Args:` and the following lines describe the arguments the function expects.
- `for student in students:`: This is a `for` loop, which is a common way to iterate through items in a list (or other iterable objects) in Python. In each iteration of the loop, the variable `student` will be assigned the next item from the `students` list.
- `print(student)`: Inside the loop, this line prints the current value of `student` (which is one of the names from the list) to the console.

The lines after the function definition demonstrate how to use the `print_students` function:

- `student_names = ["Alice", "Bob", "Charlie", "David"]`: This line creates a Python list named `student_names` containing four string elements (the student names).
- `print_students(student_names)`: This line calls the `print_students` function, passing the `student_names` list as the argument. The function then executes, printing each name in the list.

Task 5:

Write a **Python** class Car with attributes: brand, model, year. Add a **method** display_details() that prints car details.

Python:

```
▶ class Car:
    """Represents a car with brand, model, and year."""

    def __init__(self, brand, model, year):
        """Initializes the Car object with brand, model, and year."""
        self.brand = brand
        self.model = model
        self.year = year

    def display_details(self):
        """Prints the details of the car."""
        print(f"Brand: {self.brand}")
        print(f"Model: {self.model}")
        print(f"Year: {self.year}")

    # Example usage:
    my_car = Car("Toyota", "Corolla", 2020)
    my_car.display_details()

→ Brand: Toyota
Model: Corolla
Year: 2020
```

Translate the same into a **Java** class Car with attributes and a **method** displayDetails().

Java:



The screenshot shows a Java IDE interface with the following details:

- Car.java:** The file name is displayed at the top left.
- Code Content:**

```
1- public class Car {
2-     String brand;
3-     String model;
4-     int year;
5-
6-     // Constructor
7-     public Car(String brand, String model, int year) {
8-         this.brand = brand;
9-         this.model = model;
10-        this.year = year;
11-    }
12-
13-    // Method to display car details
14-    public void displayDetails() {
15-        System.out.println("Brand: " + brand);
16-        System.out.println("Model: " + model);
17-        System.out.println("Year: " + year);
18-    }
19-
20-    // Main method for example usage
21-    public static void main(String[] args) {
22-        // Create a Car object
23-        Car myCar = new Car("Toyota", "corolla", 2020);
24-
25-        // Display details of the car
26-        myCar.displayDetails();
27-    }
28-}
```
- Output Window:** Located at the bottom, showing the console output:

```
Model: corolla
Year: 2020

...Program finished with exit code 0
Press ENTER to exit console.
```

Explanation:

Explanation

◆ Certainly! The code in cell 5d9d9b53 defines a Python class called `Car` and demonstrates how to create an object (or instance) of that class and use its method.

Here's a breakdown:

- `class Car:` This line defines a new class named `Car`. Classes serve as blueprints for creating objects.
- `"""Represents a car with brand, model, and year."""`: This is a docstring explaining the purpose of the class.
- `def __init__(self, brand, model, year):`: This is a special method called the constructor. It's automatically called when you create a new `Car` object.
 - `self` refers to the instance of the class being created.
 - `brand`, `model`, and `year` are parameters that you pass when creating a `Car` object.
 - `self.brand = brand`, `self.model = model`, `self.year = year`: These lines assign the values passed as arguments to the corresponding attributes of the `Car` object. These attributes (`brand`, `model`, `year`) store the data for each specific car object.
- `def display_details(self):`: This defines a method called `display_details` within the `Car` class. Methods are functions that belong to a class and operate on the objects of that class.
 - `self` is required as the first parameter in instance methods and refers to the object the method is being called on.
 - `print(f"Brand: {self.brand}")`, `print(f"Model: {self.model}")`, `print(f"Year: {self.year}")`: These lines print the values of the `brand`, `model`, and `year` attributes of the specific `Car` object that `display_details()` is called on.

The lines after the class definition show how to use the `Car` class:

- `my_car = Car("Toyota", "Camry", 2022)`: This line creates a new object (an instance) of the `Car` class. It calls the `__init__` constructor, passing "Toyota", "Camry", and 2022 as arguments for `brand`, `model`, and `year` respectively. The created object is assigned to the variable `my_car`.
- `my_car.display_details()`: This line calls the `display_details()` method on the `my_car` object. This executes the code within the `display_details` method for the `my_car` object, printing its specific brand, model, and year.