ASSIGNMENT 19

RAMIDI . SAI CHARAN

2403A52124

TASK 1 :

PYTHON

```python
def print_numbers():
    # Loop from 1 up to (but not including) 11
    for i in range(1, 11):
        print(i)

# Call the function to display the results
print_numbers()
```

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

JAVA SCRIPT

```javascript
1 function printNumbers() {
2     // Loop from 1 up to and including 10
3     for (let i = 1; i <= 10; i++) {
4         console.log(i);
5     }
6 }
7
8 // Call the function to display the results
9 printNumbers();
```

ASSIGNMENT 19

EXPLANATION :

This Python code defines a function called `print_numbers`. Inside the function, a `for` loop iterates through numbers from 1 up to (but not including) 11, effectively covering the numbers 1 through 10. In each iteration, the current number `i` is printed. Finally, the `print_numbers()` function is called to execute the code and display the output.

TASK 2 :

PYTHON

```python
def check_number(num):
    # Check if the number is greater than zero
    if num > 0:
        print("The number is positive")
    # Check if the number is less than zero
    elif num < 0:
        print("The number is negative")
    # Otherwise, it must be zero
    else:
        print("The number is zero")

# Call the function to display the results
print("--- Python Output ---")
check_number(-5)
check_number(0)
check_number(7)

--- Python Output ---
The number is negative
The number is zero
The number is positive
```
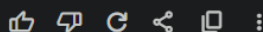
ASSIGNMENT 19

JAVASCRIPT

```java
1  public class ConditionalConverter {
2
3      /**
4       * Checks if the given number is positive, negative, or zer
5       * @param num The integer to check.
6       */
7      public static void checkNumber(int num) {
8          if (num > 0) {
9              System.out.println("The number is positive");
10         } else if (num < 0) {
11             System.out.println("The number is negative");
12         } else {
13             System.out.println("The number is zero");
14         }
15     }
16
17     public static void main(String[] args) {
18         System.out.println("--- Java Output ---");
19         // Test cases
20         checkNumber(-5); // Expected: The number is negative
21         checkNumber(0);  // Expected: The number is zero
22         checkNumber(7);  // Expected: The number is positive
23     }
24 }
```

EXPLANATION :

Here is the concise explanation of the `ConditionalConverter` Java code:

1. The `checkNumber` method uses **conditional logic** to determine the sign of the input integer (`num`).

2. It first checks if `num > 0` (positive).

3. If not, it checks if `num < 0` (**negative**) using `else if` .

4. The final `else` block handles the only remaining case: when the number is **zero**.

5. The `main` method simply executes this logic three times to demonstrate the results for negative, zero, and positive inputs.

👍 👎 ⟳ ⤓ ▢ ⋮

ASSIGNMENT 19

TASK 3 :

PYTHON

```python
def factorial(n):
    # Base Case: Factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    # Recursive Step: n * factorial(n-1)
    elif n > 1:
        return n * factorial(n - 1)
    # Handle negative numbers (optional, but good practice)
    else:
        return "Error: Factorial is not defined for negative numbers."


# Test cases
print(f"--- Python Output ---")
num1 = 5
print(f"Input: {num1} → Factorial = {factorial(num1)}")


num2 = 0
print(f"Input: {num2} → Factorial = {factorial(num2)}")


--- Python Output ---
Input: 5 → Factorial = 120
Input: 0 → Factorial = 1
```

C++

```cpp
1   #include <iostream>
2   // Function definition
3   int factorial(int n) {
4       // Base Case: Factorial of 0 or 1 is 1
5       if (n == 0 || n == 1) {
6           return 1;
7       }
8       // Handle negative numbers (optional)
9       else if (n < 0) {
10          // A simple way to signal an error; for robustness, throwing an exception is better.
11          std::cerr << "Error: Factorial is not defined for negative numbers." << std::endl;
12          return -1; // Return a sentinel value for error
13      }
14      // Recursive Step: n * factorial(n-1)
15      else {
16          return n * factorial(n - 1);
17      }
18  }
19  // Main function to test the recursive function
20  int main() {
21      std::cout << "--- C++ Output ---" << std::endl;
22
23      // Test case 1: Input 5
24      int num1 = 5;
25      int result1 = factorial(num1);
```
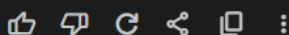
ASSIGNMENT 19

```cpp
26      if (result1 != -1) {
27          std::cout << "Input: " << num1 << " → Factorial = " << result1 << std::endl;
28      }
29
30      // Test case 2: Input 0
31      int num2 = 0;
32      int result2 = factorial(num2);
33      if (result2 != -1) {
34          std::cout << "Input: " << num2 << " → Factorial = " << result2 << std::endl;
35      }
36
37      return 0;
38  }
```

```
-- C++ Output ---
nput: 5 → Factorial = 120
nput: 0 → Factorial = 1


..Program finished with exit code 0
ress ENTER to exit console.▮
```

EXPLANATION :

Here is the six-line explanation of the recursive factorial function:

1. The function uses **recursion** to calculate $n!$, where a function calls itself.

2. The **Base Case** is $n = 0$ or $n = 1$, where the function stops and returns **1**.

3. The **Recursive Step** is $n! = n \times (n - 1)!$, reducing the problem size by 1 in each call.

4. It keeps multiplying n by the factorial of the next smaller number until it hits the base case.

5. This process then unwinds, multiplying the results back up to yield the final factorial value.

6. The logic is identical in both the dynamically typed Python and the explicitly typed C++ implementation.

TASK 4:

ASSIGNMENT 19

JAVASCRIPT

```javascript
 1  function printStudents(students) {
 2      console.log("Student List:");
 3      // Iterate over the array elements
 4      for (const name of students) {
 5          console.log(name);
 6      }
 7  }
 8  // Sample data and function call
 9  const studentArray = ["Alice", "Bob", "Charlie"];
10  printStudents(studentArray);
```

PYTHON

```python
def print_students(students):
    print("Student List:")
    # Iterate over the list elements
    for name in students:
        print(name)


# Sample data and function call
student_list = ["Alice", "Bob", "Charlie"]
print_students(student_list)

Student List:
Alice
Bob
Charlie
```

EXPLANTION :

1. Both languages define a function (`printStudents` / `print_students`) that accepts a **data structure** (Array in JS, List in Python).

2. The **JavaScript Array** (`const studentArray`) is functionally equivalent to the **Python List** (`student_list`).

3. The core logic in both is a **loop** used to iterate through the names one by one.

4. JavaScript uses a `for (const name of students)` loop, while Python uses the simpler `for name in students` loop.

5. JavaScript uses `console.log()` for output, whereas Python uses the `print()` function.

6. Both functions successfully demonstrate iterating over a sequential data structure to print all its elements.

ASSIGNMENT 19

TASK 5 :

PYTHON

```python
class Car:
    # Constructor method
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    # Method to display details
    def display_details(self):
        print("Car Details:")
        print(f"Brand: {self.brand}")
        print(f"Model: {self.model}")
        print(f"Year: {self.year}")

# 1. Create an object (instance of the Car class)
my_car_python = Car("Toyota", "Corolla", 2020)

# 2. Call the method
my_car_python.display_details()

Car Details:
Brand: Toyota
Model: Corolla
Year: 2020
```

JAVA

```java
public class Car {
    // Attributes (Instance Variables) with explicit types
    String brand;
    String model;
    int year;

    // Constructor (Must be named after the class)
    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    // Method to display details
```

ASSIGNMENT 19

```java
    public void displayDetails() {
        System.out.println("Car Details:");
        System.out.println("Brand: " + this.brand);
        System.out.println("Model: " + this.model);
        System.out.println("Year: " + this.year);
    }
}

// Main class to run the program and create the object
class Main {
    public static void main(String[] args) {
        // 1. Create an object (instance of the Car class)
        Car myCarJava = new Car("Toyota", "Corolla", 2020);

        // 2. Call the method
        myCarJava.displayDetails();
    }
}
```

EXPLANATION :

1. This exercise compares **Object-Oriented Programming (OOP)** in Python and Java through a `Car` class.

2. Both languages define a class with attributes: `brand`, `model`, and `year`, and a method to print them.

3. **Python** uses **dynamic typing**, where attributes are defined inside the special constructor `__init__`.

4. The Python method `display_details()` explicitly uses the `self` keyword to access the object's attributes.

5. **Java** is **statically typed**, requiring explicit data type declarations (e.g., `String`, `int`) for all attributes.

6. The **Java constructor** must be named the same as the class (`Car`) and uses the `new` keyword during object creation.

7. The Java method `displayDetails()` implicitly uses the `this` keyword to reference instance attributes.