1.1P: Preparing for OOP – Answer Sheet

Introduction

This paper 's answer sheet serves two purposes:

A. It serves as a revision for you of your previous learnings; and
B. It establishes a baseline understanding of your knowledge in key Computer Science topics.

As such this paper is divided into the following areas of knowledge:

A. Your experience with UNIX/DOS console commands;
B. Your ability to differentiate between data types (e.g. text) and information categories (e.g. title);
C. Your experience with compiler parsing and evaluation of expressions according to rules of precedence (e.g. BODMAS, also known as GEMS or PEMDAS);
D. Your understanding of Computer Science concepts and various compiler constructs such as blocks and scope;
E. Finally taking three steps, we want you to develop a program as follows:
    1. starting with a simple function: you provide the pure logic and calculations, no input, nor output;
    2. Then, in the second step, you write the main line code that invokes that simple function. Your main line code will provide the necessary data, and then you will print out the result of the function's calculation.
    3. Finally we want you to add business logic to the main line program's code; that business logic will interpret the results of the function, and inform your user with information about the results.

Section A: Console commands

1. Explain the following terminal instructions:
    a. cd: to **navigate between directories** in the terminal
    b. pwd: to **displays the full path of the current directory**
    c. mkdir: to **creates a new directory**
    d. cat: to **display the content of a file**
    e. ls: to **lists the files and directories** in the current directory

Section B: Data types and Information categories

1. Consider the following categories of information, and suggest the most appropriate data type to store and represent each kind of information:

Information Category	**Suggested Data Type**
A person’s family name	String
A person’s age in years	Integer
A person's weight in Kilograms	Float (or Double)
A telephone number	String
A temperature on the Kelvin scale	Float (or Double)
The average age of a group of children	Float (or Double)
Whether the student passed this task	Boolean

2. Aside from the examples already provided above, please come up with your own examples of information that could be stored as:

Data Type	**Suggested Information Category**
String	book title
Integer	number of students in a class
Float	price of a laptop in USD
Boolean	light switch is ON or OFF

Section C: Compiler evaluation of expressions

1. Fill out the **last** two columns of the following table based on the expression and values we have supplied.
2. Evaluate the value of each expression under column 1, given its formula, values, and variables; use the given values (column 2) of any variable(s) in the expression.
3. Identify the value of the results (column 3), and the data type the result is most likely to be (column 4) in a complier “friendly” form (e.g. Float):

Expression	**Given**	**Result**	**Data Type**
`76`	(NONE)	76	Integer
`True`	(NONE)	True	Boolean
`a`	`a = 3.1415927`	3.1415927	Float
`1 + 2 * 3 + 4`	(NONE)	11	Integer

`a and False`	`a = True`	False	Boolean
`a or False`	`a = True`	True	Boolean

`a + b`	`a = 1`
	`b = 3`
		4	Integer

`3 * a`	`a = 5`	15	Integer

`a * 2 + b`	`a = 2.5`
	`b = 3`
		8.0	Float

`a + 2 * b`	`a = 2.5`
	`b = 3`
		8.5	Float

`(a + b) * c`	`a = 2`
	`b = 4`
	`c = 6`
		36	Integer

`"Fred" + " Astair"`		"Fred Astair"	String
`a + " Rogers"`	`a = "Ginger"`	"Ginger Rogers"	String

Section D: Compiler Constructs and CS Concepts:

1. Using some code as an example, please explain the difference between **declaring** and **initialising** a variable.

The difference between the two is **that declaring a variable is creating it by specifying its name and type, while initializing a variable is assigning it an initial value**.

Paste your example code below:

```
// Declaration (reserves memory for an integer variable)

int z;

// Initialization (assigning an initial value)

z = 20;

// Declaration + Initialization in one step

int w = 30;
```

2. Explain the term **parameter**. Write some **code** that demonstrates a simple of use of a parameter. You should show a procedure or function that uses a parameter, and how you would call that procedure or function.

A parameter is a variable that is passed into

a function or procedure to provide input for

processing.

Paste your example code below:

```
class Program

{

    // Method with a parameter

    public static void Greet(string name)

    {

        Console.WriteLine("Hello, " + name + "!");

    }



    public static void Main()

    {

        // Calling the method with arguments

        Greet("Alice");

       Greet("Bob");

    }

}
```

3. Using an **coding example**, describe the term **scope** as it is used in procedural programming (not in business nor project management). Make sure you explain the differences of as many kinds of scope that you can identify (at least two, and up to five).

Scope is the region in a program where a variable is accessible. It defines the lifetime and visibility of a variable within a program. In procedural programming, there are different types of scope, including local, global, and block scope.

your example code below:

```
class Program
{
    // Global Scope (Accessible everywhere in this class)
    static int globalVar = 10;

    static void LocalScopeExample()
    {
        // Local Scope (Accessible only inside this function)
        int localVar = 5;
        Console.WriteLine("Local Variable: " + localVar);
    }

    static void BlockScopeExample()
    {
        if (true)
        {
            // Block Scope (Accessible only inside this block)
            int blockVar = 20;
            Console.WriteLine("Block Variable: " + blockVar);
        }
        // Console.WriteLine(blockVar); // ERROR: Not accessible here
    }

    static void Main()
    {
        Console.WriteLine("Global Variable: " + globalVar); // Accessible
here

        LocalScopeExample();
        BlockScopeExample();
    }
}
```

Section E: Implementing Algorithms, Data Handling, and Informing Results - Personalized Requirements

STEP 1:

1. In a procedural style, in any language you prefer, write a function called Average, which accepts an array of integers, and returns the average of those integers.
2. **Do not use any libraries for calculating the average**: we want to see your understanding of algorithms.
3. You must demonstrate appropriate use of parameters, returning and assigning values, and the use of loop(s). **Note — just write the function at this point.** In the next step we will ask you to ***invoke the function.***
4. You should **not** have a complete program, **nor** even code that outputs anything at this stage. This is a **function**; and input/output and any business logic processing is the responsibility of the (main line) calling code.

Paste your example function code below:

```
def average(numbers):
    total = 0  # Initialize sum variable
    count = len(numbers)  # Get the total number of elements

    for num in numbers:
        total += num

    # Return the average
    return total / count
```

STEP 2:

5. Using the same preferred language, write the main line calling code you wouldneed to (a) marshal the data, (b) invoke the function, (c) print out the result, and (d) **print out your student name and student Id**
6. We do **not** require you to provide any input processing logic; you sim ply have provide the inline instantiate of a collection of data values (provided below) for the function to calculate the average of that data set.
   a. Sample data values
      2.5, -1.4, -7.2, -11.7, -13.5, -13.5, -14.9, -15.2, -14.0, -9.7, -2.6, 2.1
7. Note: your should have made **no changes** to your function.

Paste all of your example code below:

```
# Function from the previous step
def average(numbers):
    total = 0  # Initialize sum variable
    count = len(numbers)  # Get the total number of elements

    for num in numbers:
        total += num

    # Return the average
    return total / count

# Step 5: Marshal data
data_values = [2.5, -1.4, -7.2, -11.7, -13.5, -13.5, -14.9, -15.2, -14.0, -9.7, -2.6, 2.1]

# Invoke the function
result = average(data_values)

# Print results
print("Average of data values:", result)
print("Student Name: Show Wai Yan")
print("Student ID: 105293041")
```

Paste your example code's output here:

```
Average of data values: -8.258333333333333
Student Name: Show Wai Yan
Student ID: 105293041
```

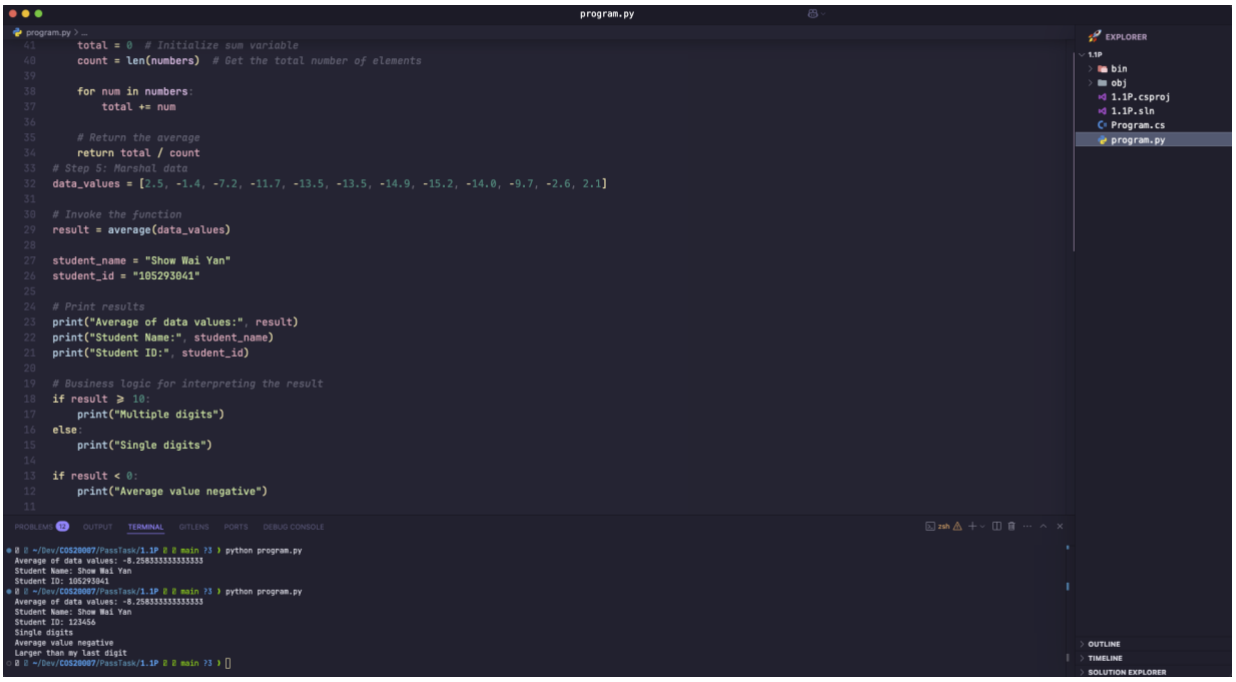
8. Using the same preferred language, add to your existing main line code above, the following business logic code for interpreting the result of the function's calculations.

9. Print the message “Multiple digits” if the average is above or equal to 10. Otherwise, print the message “Single digits”.
10. And then, if the average is negative, add an additional line of output stating “Average value negative”.
11. Finally, if the last digit of the average is larger than the last digit of your Student ID, please print the message "**Larger than my last digit**". Otherwise, please print the correct message, either "**Equal to my last digit**" or "**Smaller than my last digit**".
12. Note, you should not have made any changes to your implemented function
13. Provide evidence of your program running, i.e. the code, its environment, and its run time outputs.

Paste your example code’s output here:

```
# Function to calculate the average of an array
def average(numbers):
    total = 0  # Initialize sum variable
    count = len(numbers)  # Get the total number of elements

    for num in numbers:
        total += num

    # Return the average
    return total / count
# Step 5: Marshal data
data_values = [2.5, -1.4, -7.2, -11.7, -13.5, -13.5, -14.9, -15.2, -14.0, -9.7, -2.6, 2.1]

# Invoke the function
result = average(data_values)

student_name = "Show Wai Yan"
student_id = "105293041"

# Print results
print("Average of data values:", result)
print("Student Name:", student_name)
print("Student ID:", student_id)

# Business logic for interpreting the result
if result >= 10:
    print("Multiple digits")
else:
    print("Single digits")

if result < 0:
    print("Average value negative")

last_digit_avg = abs(int(result)) % 10  # Get last digit of absolute average value
last_digit_id = int(student_id[-1])  # Get last digit of student ID


if last_digit_avg > last_digit_id:
    print("Larger than my last digit")
elif last_digit_avg < last_digit_id:
    print("Smaller than my last digit")
else:
    print("Equal to my last digit")
```

Finally on a new page paste a SINGLE screenshot of your program (main line and function) running with its outputs here:



```python
    total = 0  # Initialize sum variable
    count = len(numbers)  # Get the total number of elements

    for num in numbers:
        total += num

    # Return the average
    return total / count
# Step 5: Marshal data
data_values = [2.5, -1.4, -7.2, -11.7, -13.5, -13.5, -14.9, -15.2, -14.0, -9.7, -2.6, 2.1]

# Invoke the function
result = average(data_values)

student_name = "Show Wai Yan"
student_id = "105293041"

# Print results
print("Average of data values:", result)
print("Student Name:", student_name)
print("Student ID:", student_id)

# Business logic for interpreting the result
if result ⩾ 10:
    print("Multiple digits")
else:
    print("Single digits")

if result < 0:
    print("Average value negative")

```

```
~/Dev/COS20007/PassTask/1.1P main ?3 ❯ python program.py
Average of data values: -8.258333333333333
Student Name: Show Wai Yan
Student ID: 105293041
~/Dev/COS20007/PassTask/1.1P main ?3 ❯ python program.py
Average of data values: -8.258333333333333
Student Name: Show Wai Yan
Student ID: 123456
Single digits
Average value negative
Larger than my last digit
~/Dev/COS20007/PassTask/1.1P main ?3 ❯ 
```

End of Task

Please render your paper as a PDF and submit via CANVAS.