1.1P: Preparing for OOP – Answer Sheet

1. Explain the following terminal instructions:
    a. cd: allow user to change between different directories.
    b. ls: list all files, folders, etc from the current directory.
    c. pwd: show the path name of the current directory.

2. Consider the following kinds of information, and suggest the most appropriate data type to store or represent each:

Information	**Suggested Data Type**
A person's name	String
A person's age in years	Integer
A phone number	String
A temperature in Celsius	Float
The average age of a group of people	Float
Whether a person has eaten lunch	Boolean

3. Aside from the examples already provided in question 2, come up with an example of information that could be stored as:

Data type	**Suggested Information**
String	Your country
Integer	This year
Float	My height
Boolean	Whether a car is red

4. Fill out the last two columns of the following table, evaluating the value of each expression and identifying the data type the value is most likely to be:

Expression	**Given**	**Value**	**Data Type**
`6`		6	integer
`True`		True	boolean

`a`	`a = 2.5`	2.5	float
`1 + 2 * 3`		7	integer
`a and False`	`a = True`	false	boolean
`a or False`	`a = True`	true	boolean
`a + b`	`a = 1` `b = 2`	3	integer
`2 * a`	`a = 3`	6	integer
`a * 2 + b`	`a = 2.5` `b = 2`	7	float
`a + 2 * b`	`a = 2.5` `b = 2`	6.5	float

`(a + b) * c`	`a = 1`
	`b = 1`
	`c = 5`
		10	integer

`"Fred" + " Smith"`		Fred Smith	string
`a + " Smith"`	`a = "Wilma"`	Wilma Smith	string

5. Using an example, explain the difference between **declaring** and **initialising** a variable.

+ **Declaring**: int x;

Declaring a variable involves telling the compiler or interpreter what type of data the variable will hold and what its name is. In essence, you're creating a "placeholder" for a value of a specific data type. This step informs the programming environment about the existence of the variable, but it doesn't allocate memory for it or assign any specific value.

+ **Initialising**: int x = 10;

Initializing a variable involves giving it an initial value. This value is assigned to the variable for the first time. Initialization can be done at the time of declaration or later in the program, depending on the programming language and the specific requirements of your code.

6. 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 which is passed to the function or method. The function or method can use them for their works.

In this Ruby example, the method calculate_rectangle_area takes two parameters, length and width. When I call the method with arguments the parameters length and width inside the method are assigned the values of the provided arguments. The method then calculates the area of the rectangle using the input values and returns the result.

```ruby
def calculate_rectangle_area(length, width)
area = length * width
return area
end

# Calling the method with specific values as arguments
length_input = 5
width_input = 3
result = calculate_rectangle_area(length_input, width_input)
puts "The area of the rectangle with length #{length_input} and width #{width_input} is #{result}"
```

7. Using an example, describe the term **scope** as it is used in procedural programming (not in business or project management). Make sure you explain the different kinds of scope.

   Scope is used to define the area that a specific variable or function can be called.
   Global scope means variable or function can be called anywhere.
   Variables and functions declared within a function have local scope, meaning they are accessible only within that function. They are not accessible from outside the function.
   In some languages, such as C#, variables declared within a block (a set of statements enclosed in curly braces) have block scope. This means they are only accessible within that block and any nested blocks.

   Using the same example from question 6,
   Global Scope: The length_input and width_input variables are declared outside of any functions or blocks. They are in the global scope and can be accessed anywhere in the code after their declaration.
   Local Scope the parameters length and width of the calculate_rectangle_area method have local scope. They are only accessible within the scope of the method.

8. In a procedural style, in any language you like, write a function called Average, which accepts an array of integers and returns the average of those integers. Do not use any libraries for calculating the average. You must demonstrate

appropriate use of parameters, returning and assigning values, and use of a loop. Note — just write the function at this point, we'll *use* it in the next task. You shouldn't have a complete program or even code that outputs anything yet at the end of this question.

1.1PResources > 1.1.rb

```ruby
def average(numbers)
  total = 0
  count = 0

  numbers.each do |num|
    total += num
    count += 1
  end

  return nil if count == 0

  average = total.to_f / count
  return average
end
```

<insert a screenshot of your code here>

9. In the same language, write the code you would need to call that function and print out the result.

```ruby
#Call the function Average
input_numbers = [11, 12, 13, 14, 15]
result = average(input_numbers)

# Printing out the result
if result.nil?
  puts "Try again. There is no appropriate number"
else
  puts "The average is: #{result}"
end
```

<insert a screenshot of your code here>

10. To the code from 9, add code to print the message "Double digits" if the average is above or equal to 10. Otherwise, print the message "Single digits". Provide a screenshot of your program running.

<insert a screenshot of your code here>

```
#question 10
if result >= 10
  puts "Double digits"
else
  puts "Single digits"
end
end
```

<insert a screenshot of your whole program running here>

1.1PResources > 1.1.rb

```ruby
def average(numbers)
  total = 0
  count = 0

  numbers.each do |num|
    total += num
    count += 1
  end

  return nil if count == 0

  average = total.to_f / count
  return average
end

#Call the function Average
input_numbers = [11, 12, 13, 14, 15]
result = average(input_numbers)

# Printing out the result
if result.nil?
  puts "Try again. There is no appropriate number"
else
  puts "The average is: #{result}"
end

#question 10
if result >= 10
  puts "Double digits"
else
  puts "Single digits"
end
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
zsh: command not found: 1.1.rb
dylantran@Dylans-MacBook-Air 1.1PResources % ruby 1.1.rb
dylantran@Dylans-MacBook-Air 1.1PResources % ruby 1.1.rb
The average of the numbers is: 13.0
dylantran@Dylans-MacBook-Air 1.1PResources % ruby 1.1.rb
1.1.rb:33: syntax error, unexpected end, expecting end-of-input
dylantran@Dylans-MacBook-Air 1.1PResources % ruby 1.1.rb
The average is: 13.0
Double digits
dylantran@Dylans-MacBook-Air 1.1PResources %
```