Homework 1

Due Date

Friday, 2/9/24, 9:00pm

Overview

Instructions

- Problems 1-4 consist of a series of code snippets for you to interpret and debug. For Problems 1-3, you will be asked to identify relevant error(s) and fix the code. For Problem 4, the code works as intended; your goal is to identify the code's purpose by following its logic.
- Problem 5 asks you to rewrite a "script" into a function, which you will then use to conduct an experiment.

Load Environment

The following code loads the environment and makes sure all needed packages are installed. This should be at the start of most Julia scripts.

```
import Pkg
Pkg.activate(@__DIR__)
Pkg.instantiate()
```

```
Activating project at `~/work/simulation-data-analysis/simulation-data-analysis/assignments
```

Problems (Total: 100 Points)

Problem 1 (20 points)

You've been tasked with writing code to identify the minimum value in an array. You cannot use a predefined function. Your colleague suggested the function below, but it does not return the minimum value.

```
function minimum(array)
    min_value = 0
    for i in 1:length(array)
        if array[i] < min_value
            min_value = array[i]
        end
    end
    return min_value
end

array_values = [89, 90, 95, 100, 100, 78, 99, 98, 100, 95]
@show minimum(array_values);
```

```
minimum(array_values) = 0
```

Problem 1.1 (10 points)

Describe the logic error.

Problem 1.2 (5 points)

Write a fixed version of the function.

Problem 1.3 (5 points)

Use your fixed function to find the minimum value of `array_values`.

Problem 2 (20 points)

Your team is trying to compute the average grade for your class, but the following code produces an error.

```
student_grades = [89, 90, 95, 100, 100, 78, 99, 98, 100, 95]
function class_average(grades)
  average_grade = mean(student_grades)
  return average_grade
end

@show average_grade;
```

```
LoadError: UndefVarError: average_grade not defined
```

Problem 2.1 (10 points)

Describe the logic and/or syntax error.

Problem 2.2 (5 points)

Write a fixed version of the code.

Problem 2.3 (5 points)

Use your fixed code to compute the average grade for the class.

Problem 3 (20 points)

Your team has collected data on the mileage of different car models. You want to calculate the average mileage per gallon (MPG) for the different cars, but your code produces the same value for all of the vehicles, which makes you suspicious.

```
function calculate_MPG((miles, gallons))
    return miles / gallons
end

car_miles =  [(334, 11), (289, 15), (306, 12), (303, 20), (350, 20), (294, 14)]

mpg = zeros(length(car_miles))

for i in 1:length(car_miles)
    miles = car_miles[1][1]
    gallon = car_miles[1][2]
    mpg[i] = calculate_MPG((miles, gallon))
```

```
end
@show mpg;
```

```
mpg = [30.363636363636363, 30.363636363636363, 30.363636363636363, 30.363636363636363, 30.363
```

Problem 3.1 (10 points)

Describe the logic error.

Problem 3.2 (5 points)

Write a fixed version of the code.

Problem 3.3 (5 points)

Use your fixed code to compute the MPGs.

Problem 4 (20 points)

You've been handed some code to analyze. The original coder was not very considerate of other potential users: the function is called `mystery_function` and there are no comments explaining the purpose of the code. It appears to take in an array and return some numbers, and you've been assured that the code works as intended.

```
function mystery_function(values)
    y = []
    for v in values
        if !(v in y)
            append!(y, v)
        end
    end
    return y
end

list_of_values = [1, 2, 3, 4, 3, 4, 2, 1]
@show mystery_function(list_of_values);
```

```
mystery_function(list_of_values) = Any[1, 2, 3, 4]
```

Problem 4.1 (10 points)

Explain the purpose of `mystery_function`.

Problem 4.2 (10 points)

Add comments to the code, explaining why and how it works. Refer to "Best Practices for Writing Code Comments", and remember that bad comments can be just as bad as no comments at all. You do not need to add comments to every line (in fact, this is very bad practice), but you should note the *purpose* of every "section" of code, and add comments explaining any code sequences that you don't immediately understand.