IE5600 – Applied Programming for Industrial Systems

AY 2023/24 Semester 1

Practical Lab 05 – Functions | Classes and Objects (I)

Section A – Tutorial Discussion

Question A1 – Applying Computational Problem Solving to Your Workplace

Recall that in Practical Lab 01 Question A3 and Practical Lab 04 Question A1, you were asked to survey your workplace and identify any one opportunity or problem in which you could apply computational problem solving to create a useful software application. As we gradually learn more advanced programming tools and techniques, they can be usefully incorporated into our existing program or use to rewrite a new program to solve the computation problem more efficiently and effectively.

For this week's discussion, please revisit Week 06's problem statement and think about how procedural programming, i.e., functions, and object-oriented programming (OOP), i.e., classes and objects, can be applied to solve the problem. Elaborate on the details as much as possible without having to write any code.

Section B – Programming Exercises

Question B1 – Reusing the Mean Calculator (Basic)

Recall that in Practical Lab 04 Question B2, you were asked to write a program that asks user to input a `list` of positive numbers (including floating point numbers), and thereafter to calculate the arithmetic mean, geometric mean and harmonic mean of the dataset.

In this question, you would be rewriting the program using the procedural programming paradigm to allow the mean calculator to be reused easily in any other future programs that you might be writing.

Perform the following tasks in order:

1) Create a Python module and name it as `mean_calculator.py`.

2) Define a function `get_number()` in `mean_calculator.py` that prompts user to input a number. If the user input an empty string, return `None` to the caller.

If the user input a non-empty string, the input data should be validated as a valid number, either integer or floating-point number. Otherwise, user should be repeatedly prompted to re-input until a valid number or empty string has been inputted.

Thereafter, the function should cast the input data into the correct data type, i.e., either `int` if the number does not contain the decimal point or `float` if otherwise, and return it to the caller.

3) Define a function `get_numbers()` in `mean_calculator.py` that uses `get_number()` to obtain a `list` of numbers from user and return to the caller. The list may contain a mix of integer and floating-point numbers. The function should stop prompting for new number and return the current `list` as soon as user has entered an empty string.

4) Define a function `calculate_arithmetic_mean(nums)` in `mean_calculator.py` that takes in a `list` of number and returns the arithmetic mean of the dataset.

5) Define a function `calculate_geometric_mean(nums)` in `mean_calculator.py` that takes in a `list` of number and returns the arithmetic mean of the dataset.

6) Define a function `calculate_harmonic_mean(nums)` in `mean_calculator.py` that takes in a `list` of number and returns the arithmetic mean of the dataset.

For task (4) to (6), you may assume that the argument is a `list` of valid numbers. If the list is empty, the functions should return `None`.

Create a new Python source file and import the `mean_calculator.py` module. Using the functions that you have defined earlier in the `mean_calculator.py` module, write a program that repeatedly prompts user to input a dataset of numbers and for each dataset, repeatedly prompts user to choose the required mean number to compute. For each valid choice, the program should print out the required mean number. The program should quit the mean calculation when user chooses to change a new dataset. The program should exit when user has no more dataset to process.

Question B2 – Cars that Drive (Intermediate)

Define a `class Car` in a new Python module `carlib` that consists of the following instance attributes:

- `make` – Make or manufacturer of a car, e.g., Toyota.
- `model` – Model of a car, e.g., Corolla.
- `num_of_door` – Number of doors that a car has, e.g., 4 or 5.
- `is_car_running` – Whether the engine of a car is currently running.
- `current_trip` – A `list` of a car's motion for the current trip with a trip defined as the time period between the starting of the engine and stopping of the engine. Each entry in the `list` is a `dict` that contains two key-value pairs of `duration_in_sec` and `distance_in_m`.

Apply the principle of encapsulation to the `class Car` by using the underscored instance attributes convention and creating the required getter/setter methods.

Define the following `class` methods that collectively represent the behaviour of a `Car`:

- `startEngine` – Set `is_car_running` to `True` and (re)initialise `current_trip` to a new empty `list`.
- `stopEngine` – Set `is_car_running` to `False`.
- `addMotion` – Add a new `dict` that contains two key-value pairs of `distance_in_m` and `duration_in_sec`. Then return the speed for this motion in km/h.
- `computeCurrentTripTravelTimeInMinute` – Return the total travel time for the current trip in min.
- `computeCurrentTripTotalDistanceInKm` – Return the total distance travelled for the current trip in km.
- `computeCurrentTripAverageSpeedInKmPerHour` – Return the average speed of the current trip in km/h by summing the total distance travelled in km and then dividing by the total travel time in hour. You can reuse `computeCurrentTripTravelTimeInMinute` and `computeCurrentTripTotalDistanceInKm`.
- `toString` – Return a string representation of the object's instance variables' values.

Create a new Python source file and import the `carlib.py` module. Using the `class Car` that you have defined earlier in the `carlib.py` module, write a program that instantiate a new `Car` object. Thereafter, add a few motions to the `Car` object and then print out the computation results. Are they correct? You can refer to a sample input/output below:

```
Toyota, Corolla, 4, False, []
Toyota, Corolla, 4, True, []
Current motion 300 m 20 sec at speed 54.000 km/h
Current motion 300 m 40 sec at speed 27.000 km/h
Current motion 300 m 60 sec at speed 18.000 km/h
Current motion 1000 m 300 sec at speed 12.000 km/h
Toyota, Corolla, 4, False, [{'distance_in_m': 300, 'duration_in_sec': 20}, {'distance_in_m': 300, 'duration_in_sec': 40}, {'distance_in_m': 300, 'duration_in_sec': 60},
{'distance_in_m': 1000, 'duration_in_sec': 300}]
Travel Time = 7.000 min
Total Distance = 1.900 km
Average Speed = 16.286 km/h
```

Question B3 – Cars that Drive (Advanced)

Examine the `class Car` that you have just written and think about the following questions:

a) Are there any alternative approach(es) to solve the car problem without the use of classes and objects? Think about these alternatives without writing any code and compare the advantages and disadvantages with reference to the use of classes and objects.

b) What other instance attributes or methods can be added to the `class Car` to make it a more realistic representation of a real-world car? Try to implement them using only the concepts and techniques that we have discussed in the lecture thus far.