Python Problem Set Worksheet 1.F CMPE 261 Large Scale Programming

Osman Emre Tutay

November 2025

Problem 1: Game Score Tracker (Global vs. Local)

In this exercise, you will manage a game's score using a global variable, while using local variables to calculate potential bonuses without affecting the actual score.

Instructions:

1. Global Setup: Create a global variable named `current_score` and initialize it to `0`.
2. Function 1 (Modifying Global): Define a function `add_points(points)`:
   - This function must accept an integer `points`.
   - It must update the global `current_score` by adding the points to it.
3. Function 2 (Local Only): Define a function `preview_bonus(points)`:
   - This function calculates what the score *would be* if a double-points bonus were applied.
   - Create a local variable named `potential_score` inside the function.
   - Logic: `potential_score = current_score + (points * 2)`.
   - Return `potential_score`.
   - **Important:** This function must **not** change the global `current_score`.

Sample Usage:

```
# 1. Initial State
print(f"Start Score: {current_score}")
# Expected: 0
# 2. Update Global Score
add_points(50)
print(f"Score after Level 1: {current_score}")
# Expected: 50

# 3. Preview Bonus (Local calculation only)
# Previewing what happens if we add 10 points with a x2 bonus
preview = preview_bonus(10)
print(f"Preview with bonus: {preview}")
# Expected: 70 (50 + 20)

# 4. Verify Global Scope
# The global score should remain 50, NOT 70
print(f"Score after preview: {current_score}")
# Expected: 50
```

Problem 2: Shopping Cart Price Calculator

Define a Python function `calculate_final_price` that computes the total cost of an item after applying a discount and adding tax.

Instructions:

- Define the function with three parameters:
  - `base_price`: The original price of the item (mandatory).
  - `discount_percent`: The percentage to deduct (default is 0).
  - `tax_rate`: The tax rate as a decimal (default is 0.10, representing 10% tax).
- Logic Order: You must apply the discount *before* calculating the tax.
- The function should return (not print) the final calculated price.

Sample Usage:

```
# Case 1: Only base price (Defaults: 0% discount, 10% tax)
# Logic: 100 + (100 * 0.10) = 110.0
print(f"Price 100: {calculate_final_price(100)}")

# Case 2: Price with Discount (Defaults: 10% tax)
# Logic: 200 - 50% = 100. Then add 10% tax -> 110.0
print(f"Price 200, 50% off: {calculate_final_price(200, 50)}")

# Case 3: Price, Discount, and Custom Tax (20% tax)
# Logic: 100 - 20% = 80. Then add 20% tax -> 96.0
print(f"Full Custom: {calculate_final_price(100, 20, 0.20)}")
```

Problem 3: Mutable vs. Immutable Function Arguments

In Python, the way variables change (or don't change) after being passed to a function depends on whether the object is mutable or immutable. This exercise demonstrates the difference using a student's grade data.

Instructions:

1. Define a function `modify_score(score_val)`:
   - Accept an integer argument.
   - Inside the function, set `score_val = 100` (attempt to overwrite it).
   - Print "Inside modify_score:" followed by the value to prove it changed locally.
2. Define a function `modify_record(grade_list)`:
   - Accept a list argument.
   - Inside the function, use the `.append(100)` method to add a new grade to the list.
   - Print "Inside modify_record:" followed by the list.
3. Main Execution:
   - Create an integer `current_score = 50`.
   - Create a list `grade_record = [50, 60, 70]`.
   - Call both functions.
   - Print the variables *after* the function calls to see which one actually changed.

Sample Usage:

```
# 1. Setup Data
current_score = 50
grade_record = [50, 60, 70]

# 2. Call Functions
print("--- Calling Functions ---")
modify_score(current_score)
modify_record(grade_record)

print("\n--- Results Outside Functions ---")

# 3. Check Immutable Integer
# Expected: 50 (The global variable did NOT change)
print(f"Final Score Variable: {current_score}")

# 4. Check Mutable List
# Expected: [50, 60, 70, 100] (The global list WAS modified)
print(f"Final Grade Record: {grade_record}")
```

Problem 4: Modularizing Unit Conversions

Create a Python package named `converters` with two modules:

- `temperature.py`: Contains a function `c_to_f(celsius)` that converts Celsius to Fahrenheit.
- `distance.py`: Contains a function `km_to_miles(km)` that converts Kilometers to Miles.

Instructions:

- After creating the package, import the modules and call each function in a main program.
- Use the conversion factor 1 km = 0.621371 miles.
- Use the formula $F = (C \times 9/5) + 32$ for temperature.

Sample Output:

```
Temperature (25C to F): 77.0
Distance (100km to Miles): 62.1371
```

Problem 5: Math and Random modules

Write a Python program that uses the `random` and `math` modules to perform the following:

- Generate two random integers between 1 and 20 representing the legs of a right triangle (a and b).
- Calculate the length of the hypotenuse using the Pythagorean theorem ($c = \sqrt{a^2 + b^2}$).
- Print the result rounded to 3 decimal places.

Sample Output:

```
Side A: 5
Side B: 12
Hypotenuse (rounded): 13.000
```

Problem 6: Implementing a Speed Converter with Static and Instance Methods

Define a Python class `SpeedConverter` with the following:

- An instance method `to_mph()` that converts a stored speed in Kilometers per Hour (km/h) to Miles per Hour (mph).
- A static method `to_kmh(mph)` that converts a speed from Miles per Hour to Kilometers per Hour.

Conversion Note: 1 mph $\approx$ 1.61 km/h

Instructions:

- Write the class so that the static method can be called without creating an instance of the class.
- The constructor (`__init__`) should accept the initial speed in km/h.

Sample Usage:

```
# Case 1: Using the instance method
car_speed = SpeedConverter(100) # Input is in km/h
print(f"100 km/h to mph: {car_speed.to_mph():.2f}")
# Expected output: approx 62.11

# Case 2: Using the static method
print(f"60 mph to km/h: {SpeedConverter.to_kmh(60):.2f}")
# Expected output: approx 96.60
```