Assignment 7

Python Functions

Instructions

For all exercises, create Python scripts as described in the exercise instructions. Save these scripts in separate files named A7En.py, where n is the exercise number. Submit all script files to Gradescope Assignment 7. For each script, be sure following the best practices discussed in class, including:

- Descriptive variable names
- Consistent variable naming convention (snake_case, camelCase, or ALLCAPS for constants)
- Module docstring that describes the script functionality and how to run it from the command line if it accepts arguments
- Block and/or inline comments describing portions of code that are not obvious. Remember, you may not be as funny or original as you think you are with your comments but if you make me laugh, good job.

Upon submission, your scripts will automatically be graded for functional correctness. Gradescope will generate a report that indicates the pass/fail result of each automated test. For failed tests, the report will describe why it failed and suggest possible sources of error in your script.

You may correct errors and resubmit your scripts as many times as you like before the due date.

Exercise 1

Write a Python script that **asks the user** to enter a temperature in degrees Fahrenheit (F), and then prints out the equivalent temperatures in degrees Celsius (C) and Kelvin (K). All three temperatures printed by the script must be rounded to 2 decimal places. Script output must be exactly as shown in the example script output below.

The temperature conversions must be implemented as functions with the following names:

`convert_fahrenheit_to_celsius()`
`convert_fahrenheit_to_kelvin()`

Each function must accept one **float** parameter that is the temperature in Fahrenheit and return the converted temperature as a **float**. The functions themselves must not round the temperatures.

Example script output:

```
PS C:\\> python A7E1.py
 Enter a temperature in Fahrenheit: 32
 32.00 F == 0.00 C == 273.15 K
```

```
PS C:\\> python A7E1.py
 Enter a temperature in Fahrenheit: 0
 0.00 F == -17.78 C == 255.37 K
```

```
PS C:\\> python A7E1.py
 Enter a temperature in Fahrenheit: -459.67
 -459.67 F == -273.15 C == 0.00 K
```

```
PS C:\\> python A7E1.py
 Enter a temperature in Fahrenheit: 85.00234
 85.00 F == 29.45 C == 302.60 K
```

Hints:

- The **input()** function always returns a string
- Use the temperature conversion formulas from here: https://www.nist.gov/pml/owm/si-units-temperature

Gradescope Submission

A Python script named `A7E1.py`

Exercise 2

Write a Python function that returns the command string needed to run a script from the command line. The function must be named `script_run_command()`, and must accept the following parameters, in the order shown, having the names in **bold** and the specified default values:

- Name of the script **file** , including the file extension (no default value)
- Path of the **directory** in which the script file is located (default value 'C:\COMP86') Note: Assume the argument does not have a trailing backslash
- Name of the **interpreter** (default value 'python')

Calling `script_run_command('script.py')` should return the value `python 'C:\COMP86\script.py'`

Here are more examples:

- `script_run_command('A2E2.ps1', 'C:\\Homework', 'pwsh')`
  `pwsh 'C:\Homework\A2E2.ps1'`
- `script_run_command('script.ps1', interpreter='pwsh')`
  `pwsh 'C:\COMP86\script.ps1'`
- `script_run_command('A7E2.py', 'D:\\COMP 86\\Assignment 7')`
  `python 'D:\COMP 86\Assignment 7\A7E2.py'`

- `script_run_command('M7A3.py', interpreter='python3', directory='.')`
  `python3 '.\M7A3.py'`

Hints:

- Use an f-string or the string concatenation operator **+** to build the script run command string.

Gradescope Submission

A Python script named `A7E2.py`

Exercise 3

Write a Python script that accepts three **command line parameters** representing hours, minutes, and seconds, and then prints the total seconds exactly as shown in the example script output below.

The total seconds calculation must be implemented as a function named `**calc_total_seconds()` that accepts hours, minutes, and seconds as integer parameters, in that order, and returns the total seconds as an integer. If any of the arguments passed to the function are not integer values, the function must return **None**.

Use a named constants called `SEC_PER_MIN = 60` in the calculations.

Example script output:

```
PS C:\> python A7E3.py 1 2 3
 1 hr 2 min 3 sec == 3723 sec
```

```
PS C:\> python A7E3.py 23 60 59
 23 hr 60 min 59 sec == 86459 sec
```

Example function calls and returns:

- `*calc_total_seconds(23, 60, 59)`
  `86459`
- `calc_total_seconds(23, 60, 'samosa')`
  `None`
- `calc_total_seconds(1, 'poutine', 3)`
  `None`

Hints:

- The `int()` function will throw an exception if its argument cannot be converted to an integer. Use `try` and `except` to catch such an exception and return `None` if any argument passed to `calc_total_seconds()` is not a valid integer. See the example on lecture slide 19.
- Don't worry about handling a `None` value returned by `calc_total_seconds()` in the `main()` function since we haven't learned how to do that yet. Your script can assume all command line parameter values

are valid integers.

Gradescope Submission

A Python script named `A7E3.py`