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.
- A main() function that executes the actual body of your script outside of the created functions
- The "Python incantation":

```
if __name__ == "__main__":
    main()
```

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')
- There is no need for error-checking the value of **file**
- There is no need for the python incantation or a main() function, this script is purely just the function. This is an example of a script being made to be reused.

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.
- You can test using either the python CLI, or by using `print(script_run_command('script.py')`

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.
- 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`

Sample Script

This script meets all the common criteria. I even made it all pretty and colour-coded for you!

```
"""
Description: This script prompts the user for the current date and time,
calculates the time passed since 2pm, December 10, 1815, and outputs the result in
a human-readable format.

Parameters: None

Author: Abe - Remember, this is my name NOT your name.  :D
"""

import datetime

# Constants
# Keep them capitalized and universal to avoid being a 'global variable' which is
a bad habit.
HOURS_PER_DAY = 24
MINUTES_PER_HOUR = 60
SECONDS_PER_MINUTE = 60

def get_current_datetime():
    """Prompt the user for the current date and time and return it as a datetime
object."""
    # User input for current date and time
    user_input = input("Please enter the current date and time (YYYY-MM-DD
HH:MM:SS): ")
    return datetime.datetime.strptime(user_input, "%Y-%m-%d %H:%M:%S")
```

```python
def calculate_time_since_ada_birth(current_datetime):
    """Calculate the time since Ada Lovelace's birth."""
    # This is a comment, the above line is a docstring - these are valid
descriptions of a function definition
    ada_birth = datetime.datetime(1815, 12, 10, 14, 0, 0)  # The format is Year,
Month, Day, Hour, Minute, Second
                                                            # Or 1400.00 (2pm) -
December 10th, 1815
    time_diff = current_datetime - ada_birth
    return time_diff

def display_result(time_diff):
    """Display the result in a human-readable format."""

    # Bonus clues live here
    # ======================
    # Note the .total_seconds() function from the object and the rest of the
calculations trickling down from there.
    seconds = int(time_diff.total_seconds())
    minutes = int(seconds // SECONDS_PER_MINUTE)
    hours = int(minutes // MINUTES_PER_HOUR )

    print(f"It's been:\n\t{hours}\t\thours\n\t{minutes}\tminutes
or\n\t{seconds}\tseconds\n...since computer programming was invented.")
    print("That occurred approximately 2pm December 10, 1815 with the birth of Ada
Lovelace.")

def main():
    current_datetime = get_current_datetime()
    time_since_ada_birth = calculate_time_since_ada_birth(current_datetime)
    display_result(time_since_ada_birth)

if __name__ == "__main__":
    main()
```

BONUS - ALL OR NOTHING

- ALL bonus objectives must be achieved in order to get any bonus marks.
- Since I made it harder, the bonus is pretty significant.
- Enjoy! Don't hesitate to ask questions.

Exercise 1

- Add error-checking. If the user enters anything that is not a number, print `"That ain't gonna fly."`

Exercise 2

- Add two more functions:
  `get_script_name()` - which will return the filename of the running script and
  `get_script_location()` - which will return the absolute path, excluding the filename of the running script.

- Neither of these functions are used in the `main()`

Exercise 3

- Make the default parameters for your function the actual hours, minutes and seconds since you started learning Scripting Fundamentals. You'll find lots of clues on how to do that in this PDF.
- You either started at 5pm September 5th or 11th, 2023 or 8am September 7th or 8th depending on your course serial.
- Yikes. Now I have to figure out how on earth to test for these.

Good luck and have fun!