Zelle 3e Chapter 6 Coding Assignment

General Instructions

My expectations for your work on coding assignment exercises will grow as we progress through the course. In addition to applying any new programming techniques that have been covered in the current chapter, I will be expecting you to follow all of the good programming practices that we have adopted in the preceding weeks. Here is a quick summary of good practices that we have covered so far:

- Include a Python Docstring that describes the intent of the program.
- Place your highest-level code in a function named *main*.
- Include a final line of code in the program that executes the *main* function.
- Follow all PEP-8 Python coding style guidelines enforced by the PyCharm Editor. For example, place two blank lines between the code making up a function and the code surrounding that function.
- Choose names for your variables that are properly descriptive.
- Define CONSTANT_VALUES and use them in place of *magic numbers*.
- Always use f-strings for string interpolation and number formatting.
- When processing items from Python lists and tuples, unpack the values into variables with meaningful variable names to avoid using indexed expressions in your code.
- Close all files before the conclusion of the program.
- Remember that your program should behave reasonably when it is not given any input. This might be the result of the user pressing enter at a console prompt. Or, it might be the result of the user providing a an input file that is empty.
- Model your solution after the code that I demonstrate in the tutorial videos.
- Make sure that your test input/output matches the sample provided.
- Create a sub-directory named *data* within your PyCharm project to hold data files.
- Remember to submit all data files with your PyCharm project – including the files that were provided as starter files to this assignment.
- All functions that are not *main()* should have descriptive, action-oriented names.
- All functions should be of reasonable size.
- All functions should have high *cohesion*, and low *coupling*.
- Remember to test your program thoroughly before submitting your work.

Exercise 1 (Regular)
Copy the program *casey_at_the_bat.py* that was provided as a starter file. Rename this new program file to *casey_at_the_bat_refactored.py*.

Refactor the code so that the program is made up of multiple functions. Each function should be properly named and should have a reasonable size. Further, functions should have high *cohesion* and low *coupling*.

As with all refactoring, the behavior of the program after refactoring should be identical to the behavior before refactoring. The following is an example of expected input/output on your console from a typical test:

```
Casey at the Bat by Ernest Thayer

The outlook wasn't brilliant for the Mudville nine that day;
the score stood four to two, with but one inning more to play.
And then when Cooney died at first, and Barrows did the same,
a sickly silence fell upon the patrons of the game.

A straggling few got up to go in deep despair. The rest
clung to that hope which springs eternal in the human breast;
they thought, if only Casey could get but a whack at that –
they'd put up even money, now, with Casey at the bat.

But Flynn preceded Casey, as did also Jimmy Blake,
and the former was a lulu and the latter was a cake,
so upon that stricken multitude grim melancholy sat,
for there seemed but little chance of Casey's getting to the bat.

But Flynn let drive a single, to the wonderment of all,
and Blake, the much despised, tore the cover off the ball;
and when the dust had lifted, and the men saw what had occurred,
there was Jimmy safe at second and Flynn a-hugging third.

Then from five thousand throats and more there rose a lusty yell;
it rumbled through the valley, it rattled in the dell;
it knocked upon the mountain and recoiled upon the flat,
for Casey, mighty Casey, was advancing to the bat.

There was ease in Casey's manner as he stepped into his place;
there was pride in Casey's bearing and a smile on Casey's face.
And when, responding to the cheers, he lightly doffed his hat,
no stranger in the crowd could doubt 'twas Casey at the bat.

Ten thousand eyes were on him as he rubbed his hands with dirt;
five thousand tongues applauded when he wiped them on his shirt.
Then while the writhing pitcher ground the ball into his hip,
defiance gleamed in Casey's eye, a sneer curled Casey's lip.

And now the leather-covered sphere came hurtling through the air,
```

and Casey stood a-watching it in haughty grandeur there.
Close by the sturdy batsman the ball unheeded sped—
"That ain't my style," said Casey. "Strike one," the umpire said.

From the benches, black with people, there went up a muffled roar,
like the beating of the storm-waves on a stern and distant shore.
"Kill him! Kill the umpire!" shouted someone on the stand;
and it's likely they'd have killed him had not Casey raised his hand.

With a smile of Christian charity great Casey's visage shone;
he stilled the rising tumult; he bade the game go on;
he signaled to the pitcher, and once more the spheroid flew;
but Casey still ignored it, and the umpire said: "Strike two."

"Fraud!" cried the maddened thousands, and Echo answered fraud;
but one scornful look from Casey and the audience was awed.
They saw his face grow stern and cold, they saw his muscles strain,
and they knew that Casey wouldn't let that ball go by again.

The sneer is gone from Casey's lip, his teeth are clenched in hate;
he pounds with cruel violence his bat upon the plate.
And now the pitcher holds the ball, and now he lets it go,
and now the air is shattered by the force of Casey's blow.

Oh, somewhere in this favored land the sun is shining bright;
the band is playing somewhere, and somewhere hearts are light,
and somewhere men are laughing, and somewhere children shout;
but there is no joy in Mudville — mighty Casey has struck out.

Exercise 2 (Regular)

Copy the program *when_im_gone.py* that was provided as a starter file. Rename this new program file to *when_im_gone_refactored.py*.

Refactor the code so that the program is made up of multiple functions. Identify any duplicate code in the program and extract it into common functions. Make sure that all functions are properly named and have a reasonable size. Further, make sure that all functions have high *cohesion* and low *coupling*.

As with all refactoring, the behavior of the program after refactoring should be identical to the behavior before refactoring. The following is an example of expected input/output on your console from a typical test:

```
When I'm Gone by Phil Ochs

There’s no place in this world where I’ll belong when I’m gone
And I won’t know the right from the wrong when I’m gone
And you won’t find me singin’ on this song when I’m gone
So I guess I’ll have to do it while I’m here

And I won’t feel the flowing of the time when I’m gone
All the pleasures of love will not be mine when I’m gone
My pen won’t pour a lyric line when I’m gone
So I guess I’ll have to do it while I’m here

And I won’t breathe the bracing air when I’m gone
And I can’t even worry ’bout my cares when I’m gone
Won’t be asked to do my share when I’m gone
So I guess I’ll have to do it while I’m here

And I won’t be running from the rain when I’m gone
And I can’t even suffer from the pain when I’m gone
Can’t say who’s to praise and who’s to blame when I’m gone
So I guess I’ll have to do it while I’m here

Won’t see the golden of the sun when I’m gone
And the evenings and the mornings will be one when I’m gone
Can’t be singing louder than the guns when I’m gone
So I guess I’ll have to do it while I’m here

All my days won’t be dances of delight when I’m gone
And the sands will be shifting from my sight when I’m gone
Can’t add my name into the fight while I’m gone
So I guess I’ll have to do it while I’m here

And I won’t be laughing at the lies when I’m gone
And I can’t question how or when or why when I’m gone
Can’t live proud enough to die when I’m gone
So I guess I’ll have to do it while I’m here
```

There’s no place in this world where I’ll belong when I’m gone
And I won’t know the right from the wrong when I’m gone
And you won’t find me singin’ on this song when I’m gone
So I guess I’ll have to do it, I guess I’ll have to do it
Guess I’ll have to do it while I’m here

Exercise 3 (Regular)
Copy the program *ninety_nine.py* that was provided as a starter file. Rename this new program file to *ninety_nine_refactored.py*.

This program prints the lyrics to the tradition song *99 Bottles of Beer*. I chose this song because it is so predictably repetitive.

As part of your program, create a function named *write_numbered_verse()* that accepts an int parameter (*bottle_number)* and prints a typical numbered verse like:

37 bottles of beer on the wall, 37 bottles of beer.
Take one down and pass it around, 36 bottles of beer on the wall.

Your program should use a FOR-IN loop that counts backwards. For tips on how to do this, you can consult the following source:

- https://realpython.com/python-range/ - count-backward-with-negative-steps

Make sure that all functions in your refactored program are properly named and have a reasonable size. Further, make sure that all functions have high *cohesion* and low *coupling*.

As with all refactoring, the behavior of the program after refactoring should be identical to the behavior before refactoring. The following is an example of expected input/output on your console from a typical test. Note that quite a few lines have been omitted for space considerations:

```
99 Bottles of Beer
Traditional

99 bottles of beer on the wall, 99 bottles of beer.
Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.
Take one down and pass it around, 97 bottles of beer on the wall.

97 bottles of beer on the wall, 97 bottles of beer.
Take one down and pass it around, 96 bottles of beer on the wall.
.
.
.
```
(MANY LINES HAVE BEEN OMITTED HERE FOR SPACE CONSIDERATIONS)
```
.
.
.
```

6 bottles of beer on the wall, 6 bottles of beer.
Take one down and pass it around, 5 bottles of beer on the wall.

5 bottles of beer on the wall, 5 bottles of beer.
Take one down and pass it around, 4 bottles of beer on the wall.

4 bottles of beer on the wall, 4 bottles of beer.
Take one down and pass it around, 3 bottles of beer on the wall.

3 bottles of beer on the wall, 3 bottles of beer.
Take one down and pass it around, 2 bottles of beer on the wall.

2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottles of beer on the wall.

1 bottles of beer on the wall, 1 bottles of beer.
Take one down and pass it around, 0 bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 99 bottles of beer on the wall.

Exercise 4 (Regular)
Copy the program *abraham_martin_and_john.py* that was provided as a starter file. Rename this new program file to *abraham_martin_and_john_refactored.py*.

This program uses a data structure to hold facts about four key figures mentioned in the song *Abraham, Martin and John* by Dick Holler. For each key figure, the following values are included:

- Name
- Date of birth
- Date of death

The facts for each individual are contained in a tuple. The tuples for all of the individuals are contained in a list. Although this may be the first time you are seeing this type of data structure, this is a common method of arranging data facts within a program.

The program is complete and fully functional. Your mission is to refactor the program so that it hides the assumptions regarding how to format dates in a separate function. This will follow the *hiding places* use case for functions.

The new function that you create should be properly named and should have a reasonable size. Further, it should have high *cohesion* and low *coupling*.

As with all refactoring, the behavior of the program after refactoring should be identical to the behavior before refactoring:

```
Key Figures from the Song "Abraham, Martin and John" by Dick Holler:

Abraham Lincoln
Born: 02/12/1809
Died: 04/15/1865

Martin Luther King Jr.
Born: 01/15/1929
Died: 04/04/1968

John F. Kennedy
Born: 05/29/1917
Died: 11/22/1963

Robert F. Kennedy
Born: 11/20/1925
Died: 06/06/1968
```

Exercise 5 (Challenge)
Copy the program *abraham_martin_and_john_refactored.py* that you created in Exercise 4. Rename this new program file to *abraham_martin_and_john_pretty_dates.py*.

Change the behavior of this new program so that it prints the dates in a prettier format as shown below:

```
Key Figures from the Song "Abraham, Martin and John" by Dick Holler:

Abraham Lincoln
Born: Sunday, February 12, 1809
Died: Saturday, April 15, 1865

Martin Luther King Jr.
Born: Tuesday, January 15, 1929
Died: Thursday, April 04, 1968

John F. Kennedy
Born: Tuesday, May 29, 1917
Died: Friday, November 22, 1963

Robert F. Kennedy
Born: Friday, November 20, 1925
Died: Thursday, June 06, 1968
```

Your program should include a revised version of the function that formats dates. Here are some hints and resources for your reference when creating the new code:

1. Remember that the date values in our program are stored in ISO Format (see https://en.wikipedia.org/wiki/ISO_8601).

2. The date functionality that you require is most easily implemented using the features of the Python *datetime* package.

3. The official Python documentation for the *datetime* package (which can be difficult to navigate) is located here: https://docs.python.org/3.11/library/datetime.html

4. Your code will need to accomplish two things: create a date object using the *date* class and use the formatting method of that date object to create a formatted date string that meets your needs.

   i) Create a date object (see *date.fromisoformat()* method).
   ii) Use the formatting method of the date object to create a date string that meets your needs (see *date.strfmtime()* method).

Tools
Use PyCharm to create and test all Python programs.

Submission Method
Follow the process that I demonstrated in the tutorial video on submitting your work. This involves:

- Locating the properly named directory associated with your project in the file system.
- Compressing that directory into a single .ZIP file using a utility program.
- Submitting the properly named zip file to the submission activity for this assignment.

File and Directory Naming
Please name your Python program files as instructed in each exercise. Please use the following naming scheme for naming your PyCharm project:

surname_givenname_exercises_zelle_3e_chapter_06

If this were my own project, I would name my PyCharm project as follows:

trainor_kevin_exercises_zelle_3e_chapter_06

Use a zip utility to create one zip file that contain the PyCharm project directory. The zip file should be named according to the following scheme:

surname_givenname_exercises_zelle_3e_chapter_06.zip

If this were my own project, I would name the zip file as follows:

trainor_kevin_exercises_zelle_3e_chapter_06.zip

Due By
Please submit this assignment by the date and time shown in the Weekly Schedule.

Last Revised
2025-02-10