Certification Exam for Python Part I

Instructions

This is an open book test. You can use the examples we made in class and the Python documentation found at: https://docs.python.org/3/ .   That said, this should be you own work. It would be unfair to post a question on StackOverflow or to have an associate help out with a question.

The purpose of the test is to show off the range of your Python skills, so there is a lot of room for partial credit. If you're asked to write a class and can't remember how, then write simple functions or a plain script to do the task. If you get stuck writing a complex function, then mock up the function by returning a constant and continue with the rest of the exercise.

The other purpose of the test is to have fun applying and practicing your Python skills. It is a vehicle for cementing skills you've learned and pointing out opportunities for growth.

Type annotations are optional but will earn extra credit. Do pay attention to good variable names, function names, parameter names, and docstrings. Also pay attention to nicely formatted, readable code.

Creating reusable functions, classes, and modules was a central goal for the course, so make as many of those as possible in your answers. When there is a standard library function available, you should prefer that over rewriting your own version (unless yours is better in some way). Likewise, you should prefer the looping idioms covered in class (reversed, enumerate, etc.) Put in error handling as needed.

When you're ready, zip-up a directory containing your working code, the input data, output files, and any supporting py files, such as *beck.py*. Email your results to rachel@mutableminds.com

So that this exam can be used again for future classes, please do not share it with your teammates.

If desired, anyone passing this test is also ready to sit for a Microsoft certification.
https://docs.microsoft.com/en-us/learn/certifications/exams/98-381

Resistor Problem

Use the *beck.py* module developed in class to apply Test Driven Development (TDD) to building a useful class to encapsulate the behavior of resistors in circuits. Turn in three files: *resistor.py* containing the class, *test_resistor.py* that tests the class using *beck.py*, and *resistor_problem.py* that applies your tools to compute the answer to the resistor problem given in part 3 and part 4 below.

1) Start by supporting these basic capabilities:

```
>>> r1 = Resistor(100)
>>> vars(r1)
{'x': 100}
>>> r1
Resistor(100)
>>> print(r1)
100.0Ω
>>> r1.current(voltage=50)      # V = IR, current is 50 / 100 amps
0.5
>>> r2 = Resistor(200)
>>> r1.series(r2)               # Resistors in series add together:  100 + 200 ohms
Resistor(300)
>>> r1.parallel(r2)             # Resistors in parallel: 1 / (1 / 100 + 1 / 200) ohms
Resistor(66.66666666666667)
```

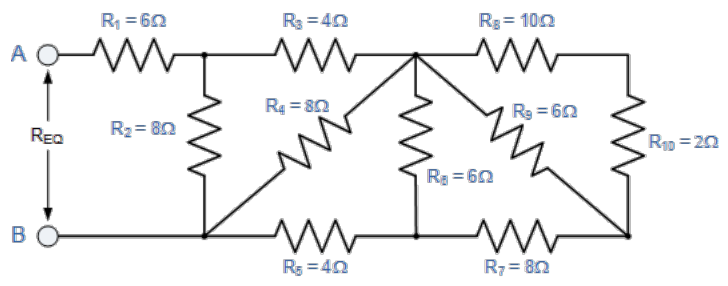
2) Add the appropriate "dunder methods" to make the API easier to work with:

```
>>> r1 + r2                    # Use the "+" operator to mean "series"
Resistor(300)
>>> r1 * r2                    # Use the "*" operator to mean "parallel"
Resistor(66.66666666666667)
>> r1 == Resistor(100)         # Define the "==" operator with __eq__()
True
```

3) Verify ability to compute more complex networks

```
>>>print((Resistor(8).series(Resistor(4))).parallel(Resistor(12)).series(Resistor(6)))
12.0Ω
>>> (Resistor(8) + Resistor(4)) * Resistor(12) + Resistor(6) == Resistor(12)
True
```

4) Use your tool to find the R_{EQ} total resistance of this network and to compute the current through it given a 12 volt source across A and B.



Reference: https://www.electronics-tutorials.ws/resistor/res_5.html

On-line dictionary problem

Access the REST API defined at https://dictionaryapi.dev/ with a reusable function that does a case insensitive lookup for a word. For each sense of the word, show the word, the part of speech, and its definition. Optionally, list synonyms if requested and available:

```
>>> define('Python')

Sense #1:
python (noun): a large heavy-bodied non-venomous snake occurring throughout the Old
World tropics, killing prey by constriction and asphyxiation.
python (noun): a high-level general-purpose programming language.

>>> define('FORTE', include_synonyms=True)

Sense #1:
forte (noun): a thing at which someone excels.
forte (synonyms): strength, strong point, speciality, long suit, strong suit, talent,
special ability, skill, bent, gift, claim to fame, department, métier, pièce de
résistance, bag, thing, cup of tea

forte (noun): the part of a sword blade from the hilt to the middle.

Sense #2:
forte (adverb): (especially as a direction) loudly.
forte (adjective): played loudly.
forte (noun): a passage performed or marked to be performed loudly.

>>> define('whiff', include_synonyms=True)

Sense #1:
whiff (noun): a smell that is only smelt briefly or faintly.
whiff (synonyms): faint smell, brief smell, trace, sniff, scent, odour, aroma

whiff (noun): a puff or breath of air or smoke.
whiff (synonyms): puff, gust, blast, rush, flurry, gale, breath, draught, waft

whiff (noun): (chiefly in baseball or golf) an unsuccessful attempt to hit the ball.
whiff (verb): get a brief or faint smell of.
whiff (verb): (chiefly in baseball or golf) try unsuccessfully to hit the ball.

Sense #2:
whiff (noun): another term for megrim2.

Sense #1:
program (noun): a set of related measures or activities with a particular long-term
aim.
program (noun): a series of coded software instructions to control the operation of a
computer or other machine.
program (noun): a presentation or item on television or radio, especially one
broadcast regularly between stated times.
program (noun): a sheet or booklet giving details of items or performers at an event
or performance.
program (verb): provide (a computer or other machine) with coded instructions for the
automatic performance of a task.
program (verb): arrange according to a plan or schedule.
program (verb): broadcast (an item).
```

Texas Freeze

The data below contains temperature measurements in Austin Texas during last year's near failure of the Texas power grid. The data comes from: https://www.accuweather.com/en/us/austin/78701/february-weather/351193

1) Make a reusable function that parse the following column-oriented text using slicing techniques. Be sure to use the headers to locate the relevant slice indices. The result should be a list of named tuples in the form, `Temperature(day: int, high: int, low: int)`.

```
data = '''\
AccuWeather
February 2021
Austin, TX
Daily Temperature History

Day   High   Low
---   ----   ---
1      67°    41°
2      69°    40°
3      77°    45°
4      85°    55°
5      61°    44°
6      75°    48°
7      70°    40°
8      76°    51°
9      64°    46°
10     46°    37°
11     37°    31°
12     33°    29°
13     31°    26°
14     30°    13°
15     25°     8°
16     26°     7°
17     32°    24°
18     33°    24°
19     43°    21°
20     62°    26°
21     76°    44°
22     76°    42°
23     79°    47°
24     86°    64°
25     66°    51°
26     60°    47°
27     79°    58°
28     74°    58°
'''
```

2) Take advantage of the *statistics* module and list comprehensions to compute:

- Mean and standard deviation for the high temperatures.
- Number of days where the low was at or below freezing.
- Minimum, three quartiles, and maximum for the lows.
- High on the first day and high on the last day.
- Maximum difference between the high and low each day.

Car Evaluation Dataset

Fundamentally, this task involves converting a CSV flat file into a hierarchical XML dataset. The task can be done quickly using the *csv module* and the *add_element* function we wrote in class or the *SubElement* class found in the *ElementTree* module.

The data set has six attributes for 1,728 vehicles: purchase price, maintenance cost, number of doors, maximum number of passengers, trunk capacity, and a safety rating. Each row also containers an overall evaluation of quality: very good, good, acceptable, and unacceptable.

The dataset can be downloaded from *car.data* at https://archive.ics.uci.edu/ml/machine-learning-databases/car/ . The meaning of each the fields is described in *car.names* also found at https://archive.ics.uci.edu/ml/machine-learning-databases/car/ .

Given input lines such as:

```
vhigh,vhigh,2,4,small,high,unacc
vhigh,low,3,more,small,high,acc
```

Translate the field values to be more human readable. Organize the XML as follows and write it to a file, *car_evaluation.xml*:

```
<?xml version='1.0' encoding='us-ascii'?>
<evaluation_set year="1990"
source="https://archive.ics.uci.edu/ml/datasets/Car+Evaluation">
<car overall_rating="Unacceptable">
<price>
<purchase_price>Very high</purchase_price>
<maintenance_cost>Very high</purchase_price>
</price>
<tech>
<comfort>
<doors>2</doors>
<passengers>4</passengers>
<trunk_capacity>Small</trunk_capacity>
</comfort>
<safety>High</safety>
</tech>
</car>
<car overall_rating="Acceptable">
<price>
<purchase_price>Very high</purchase_price>
<maintenance_cost>Low</purchase_price>
</price>
<tech>
<comfort>
<doors>3</doors>
<passengers>More than 4</passengers>
<trunk_capacity>Small</trunk_capacity>
</comfort>
<safety>High</safety>
</tech>
</car>
</evaluation_set>
```