The finals this quarter will likely be a mix of code reading and code writing. It is unlikely that you will get a solely conceptual question, but it may appear as a part of a different question.

Here is a list of topics and which questions match up with them. Topics are expected to overlap for many questions, even if a question is not listed for all of them. Additionally, the curriculum and testing format changes quarter to quarter so just because a topic has a lot of questions associated with it does not mean that it will show up on the final and vice versa. It may be helpful to look at the more recent (higher number) questions as these are more representative of the current quarter.

- Error, debugging: 1, 3, 5, 6, 9, 16, 23, 26, 30, 31, 32, 42, 47
- Variables, types, statements, expressions: 2, 8
- Loops, conditionals: 12, 15, 17, 18, 24
- Functions: 4, 7, 9, 33, 35, 36, 37, 39, 40, 41, 44, 46, 48, 49
- Data structures (lists, dictionaries, sets): 10, 16, 19, 33, 35, 36, 44, 49
- File I/O: 13, 37, 39, 45, 48
- Classes: 14, 20, 28, 34, 38, 43, 50
- Graphs: 21, 46
- Good programming practice, style: 11, 22, 25, 29
- Miscellaneous: 27 (list comprehension),

Conceptual Questions

1. Write code or an expression that would produce a "`KeyError`" error

---

2. a. Give two examples of a mutable type:

b. Give two examples of an immutable type:

---

3. Write code or an expression that would produce an `IndexError`.

---

4. Give three advantages of using functions. Your reasons should be as different as possible, and no longer than a sentence each (a phrase each is fine).

---

5. Write code that would produce an "`TypeError`" error.

---

6. When should you write tests: before, during, or after writing the code? Why?

---

7. What is Data Abstraction and why is it useful? Be specific.

---

8. For each row in the following table evaluate the expressions in the first column, all of which result in a value being assigned to the variable x. In the **type(x)** column, write what the type of that value is. In the last column, write the value that we'd see if we ran **print(x)**, i.e., show what value gets stored in **x**. If there is an error, write **Error** in both columns.

Expression	**type(x)**	**What does x evaluate to?**
`x = ("Hello" + "World") * 2`		

`x = print("not an integer?")`		
`x = "one" + "one" == "two"`		
`x = 3 * 3 / 3`		
`x = sorted("python")`		
`x = "python"[::-1]`		

`a = {'a', 'b', 'c'}`
`z = {'x', 'y', 'z'}`
`x = a & z`		

`d = {"cat": ["meow", "purr"],`
`"dog": ["bark", "woof"]}`
`l = ["cat", "dog"]`
`x = d[l[-1]][-1] + d[l[0]][0]`		

Reading Code

9. You receive the following error messages after running your code:

```
Traceback (most recent call last):
  File "social_network.py", line 329, in <module>
    do_stuff(friend_list)
  File "social_network.py", line 163, in do_stuff
    result = recommend_by_influence(friend_list)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "social_network.py", line 229, in recommend_by_influence
    output = read_result()
             ^^^^^^^^^^^^
NameError: name 'read_result' is not defined
```

a. What is the last function that was successfully called?

b. Describe how you would go about trying to find the cause of and fix this error.

---

10. Given the function sum_roll() which returns the sum of two random 6-sided dice rolls, complete the following function to create a dictionary of sum frequencies. The dictionary should contain an entry for every possible sum of two dice.

For example, freq_of_sums(5) might return: {2:1, 3:0, 4:0, 5:1, 6:0, 7:2, 8:0, 9:0, 10:1, 11:0, 12:0}

```
def freq_of_sums(n):
    '''
    Rolls two 6-sided dice n times and records the sum of each pair.
Assumes
    n is a positive integer. Returns a dictionary mapping the sums to
the
    number of times in n rolls that sum was rolled.
    '''
    # Your code here
```

a. Write a test for this function.

---

11. Write a docstring for the following function. Document the inputs and any outputs or side effects (meaning, what else does the function do?).

```
import matplotlib.pyplot as plt


def my_plot(slope, x_points):
    '''
    Your docstring would go here.
    '''
    y_points = []
    for x in x_points:
        y_points.append(x**slope)
    plt.plot(x_points, y_points)
    plt.show()
```

a. Add one line of code to `my_plot` that would save the resulting figure as “using_val.png”, where val is whatever value is stored in the variable `slope`. Draw arrows in the code above to show where you would add your line of code, be exact!

---

12. Write the output of the code below:

```
sum = 0
for x in range(2, 8):
    if x % 2 == 0:
        for y in range(x):
            sum = sum + y
print('sum:', sum)
```

13.a. Write a function called `read_zoo_animals(filename)` that takes the name of a file as a parameter. You can assume that the given file contains lines of the form: `zoo_name animal` where each `zoo_name` and `animal` are strings that contain no spaces or punctuation and are separated by a single space. Here are the contents of a sample input file:

```
point_defiance monkey
national panda
woodland_park bear
national elephant
national panda
```

Your function should read in the given file and return a dictionary mapping each `zoo_name` to a list of the animals the zoo has. For a given zoo, the animals in that zoo's list should appear in the order they appear in the file. It is o.k. to have duplicate animals in the list for a given zoo. You may assume the file name provided is valid and that the file is formatted as described and includes at least one `zoo_name animal` pair.

Calling read_zoo_animals on the file above would return this dictionary:

```
{
    'point_defiance': ['monkey'],
    'national': ['panda', 'elephant', 'panda'],
    'woodland_park': ['bear']
}
```

```
def read_zoo_animals(filename):
    # Your code here
```

b. Write code in the main function that will call the `read_zoo_animals` function written in part (a) to open a file called "`local_zoos.txt`" and print the results in EXACTLY the following format. For the sample input shown in part (a) the output would be:

```
point_defiance:
 monkey
national:
 elephant
 panda
 panda
woodland_park:
 bear
```

`zoo_names` may be printed in any order, but each zoo's list of animals should be printed in **alphabetical order**. Don't forget the colon at the end of each `zoo_name` and a single space at the beginning of each `animal`.

---

14. Note: THIS PROBLEM IS NOT RELATED TO THE "Zoo Animals" PROBLEM!!

You are given the following class definition:

```
class Zoo:
    def __init__(self, zoo_name):
        '''
        zoo_name: a string representing name of zoo
        '''
```

```
        self.name = zoo_name
        self.animals = []

    def add(self, animal):
        '''
        animal: a string representing an animal
        '''
        self.animals.append(animal)
```

a. Write the code for the method below that is also a part of the class `Zoo`:

```
    def release_animal(self, free_animal):
        '''
        Remove all occurrences of free_animal from this zoo.

        Arguments:
            free_animal: a string representing an animal.

        Returns: the number of occurrences that were removed.

        The free_animal string must match exactly (e.g. same case)
with the
        name of an animal currently in the zoo in order for that
animal to
        be removed.
        '''
        # Your code here
```

b. Write the code for the method below that is also a part of the class `Zoo`:

```
    def get_num_animals(self):
        '''
        Returns the total number of animals in the zoo as an integer.
        '''
        # Your code here:
```

c. Write code in the `main` function to add a lion and a tiger to the `seattle_zoo`. This code is outside of the class Zoo.

```
def main():
    seattle_zoo = Zoo("woodland_park")
    # Your code here:
```

d. Describe a change to the Zoo class that might cause a client of the Zoo class to have to modify its code.

e. Describe a change to the Zoo class that would NOT cause a client of the Zoo class to have to modify its code.

---

15. Write the output of the code below.

```
sum = 0
for x in range(1, 25, 2):
    temp = (x / 10) % 10
    sum = sum + temp
print('sum:', sum)
```

---

16. Indicate (circle) if there is an Error or No Error. **For any credit, if there is an error, you MUST very briefly say what the problem is.

a)

```
d = {}
d[{0, 1, 2}] = {0, 1, 2}
```

If an Error, describe briefly:

b)

```
d = {}
d[[0, 1, 2]] = {0, 1, 2}
```

If an Error, describe briefly:

c)

```
d = {}
d["0, 1, 2"] = (0, 1, 2) d["0, 1, 2"][1] = 0
```

If an Error, describe briefly:

d)

```
d = {}
d["0, 1, 2"] = {0, 1, 2}
print d.keys()["0, 1, 2"][0]
```

If an Error, describe briefly:

---

17. Write the output of the code below.

```
def mystery():
    result = 0
    for i in range(20, 0, -2):
        result += i
        if i % 8 == 0:
            result -= 8
        elif i % 4 == 0:
            result -= 4
        if i % 9 == 0:
            return result
    return result

print(mystery())
```

---

18. For each of the blanks (marked by "# ____") at the end of the lines of code below, write a boolean expression that defines what values would need to be passed into the function (as the variable x) in order for the matching line of code to be reached. You can assume that the values passed in as x will always be integers. Feel free to use mathematical notation. If a branch is unreachable, write UR. For example:

```
def example(x):
     x = x + 1
     if x > 3 or x < 3:
          print('Branch 1 reached!')          # x != 2
     elif x == 2:
          print('Branch 2 reached!')          # UR


### Part A:
def part_a(x):
     x = x * 2
     if x > 0:
          print('Branch 1 reached!')
     elif x > 1:
          print('Branch 2 reached!')
     else:
```

```
            print('Branch 3 reached!')

### Part B:
def part_b(x):
      x = x % 2
      if x > 1:
            print('Branch 1 reached!')
      elif x < 0:
            print('Branch 2 reached!')
      else:
            print('Branch 3 reached!')

### Part C:
def part_c(x):
      y = x % 5
      if y > 3:
            print('Branch 1 reached!')
            if x < 0:
                  print('Branch 1.5 reached!')
            elif x == 0:
                  print('Branch 1.75 reached!')
      else:
            print('Branch 2 reached!')
            z = x / 5
            if 5 * z == x:
                  print('Branch 2.5 reached!')
            elif y == 0:
                  print('Branch 2.75 reached!')
```

---

19. For each nested data structure, write the one line of code that would print the string "Dog". You can assume that "Dog" will only appear once. You must use the given data structure; writing `print("Dog")` will be marked as 0 points.

```
nested_data_1 = {"Canine": ["Puppy", "Hound", "Dog"],
"Feline": ["Kitten", "Cat"], "Swine":["Hog", "Pig"]}

nested_data_2 = {"Johnsons": {"Butterscotch": "Cat"},
"Smiths": {}, "Garcias": {"Guppy": "Fish", "Scout": "Dog"}}

nested_data_3 = [{"France": {"Population": 638000, "Area":
9870, "Pet": "Fish"}}, {"United States": {"Population":
323000000, "Area": 5870000, "Pet": "Dog"}}, {"Russia":
```

```
{"Population": 143000000, "Area": 66000000, "Pet": "Cat"}}]

nested_data_4 = [["Building", "Cars", "Lights", "Loud"],
["Dog", "Cow", "Sheep", "Tractor", "Field"], ["Boat",
"Fishing", "Creek", "Quiet"]]
```

---

20. Consider the following class definition:

```
from operator import itemgetter
class RestaurantRating:
    def __init__(self, name):
        self.rating = {}
        print(name + "'s top restaurants list")

    def add_rating(self, restaurant_name, rating):
        self.rating[restaurant_name] = rating

    def get_rating(self):
        r1 = sorted(self.rating.items())
    r2 = sorted(r1, key=itemgetter(1), reverse=True)
        final = []
        for pair in r2:
            final.append(pair[0])
        return final
```

For each of the below code snippets, write the expected output.

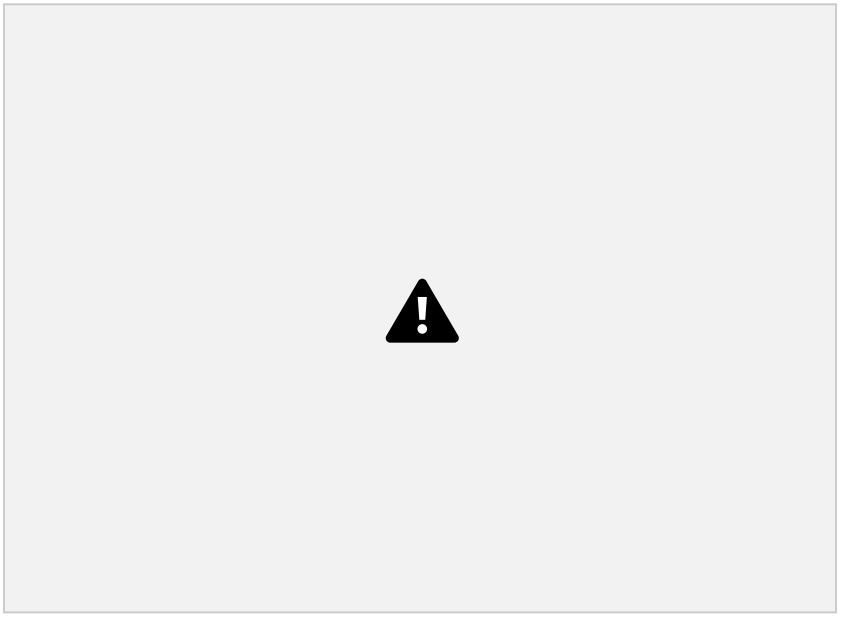
```
# Snippet A
rating1 = RestaurantRating("The Ave")
rating1.add_rating("Chi-Mac", 4.0)
rating1.add_rating("Cafe on the Ave", 4.3)
rating1.add_rating("Korean Tofu House", 4.3)
print(rating1.get_rating())
# Write Snippet A's output here:
```

```
# Snippet B
rating2 = RestaurantRating("UVillage")
rating2.add_rating("Dough Zone", 4.0)
rating2.add_rating("Din Tai Fung", 4.0)
rating2.add_rating("Elemental", 4.2)
print(rating2.get_rating())
```
 # Write Snippet B's output here:

---

21. The data file called "Building_names.txt" has been provided to you, and contains pairs of building name abbreviations. This file contains:

```
KNE MGH
CLB MGH
THO MGH
HUB MGH
HUB THO
```



You may assume that there are only two columns like above, separated by four spaces ("whitespace"). Each line of the file represents an edge in a graph. In the space provided below, write the code to read the file, create a graph, and then plot and show the graph. "`networkx`" and "`matplotlib.pyplot`" are imported for you below. Make sure to use `.draw_networkx(graph)` to draw the graph and `.show()` to show the graph.
The resulting graph should look like what's shown to the right.

```
import networkx as nx
import matplotlib.pyplot as plt
```

---

22. Your friend wants you to see the code that they've written. While you look at their code, you start having flashbacks of the flake8 style errors that you lost in your homework submissions for CSE 160. Given this code, circle four general style errors (not just those that flake8 would catch). Then for each style error, write a sentence (also noting the line number) explaining why it is an error (i.e., which style guideline is it violating?) and provide a possible solution.

Line Numbers

```
1
    myFavoriteSongs = ["OMG", "Rocketeer", "Clarity", "Plastic Love", "Jam &
Butterfly"] 2
    count = 0
3
    for i in range(0,len(myFavoriteSongs),1):
4
        x = myFavoriteSongs[i]
5
        if (len(x) > 7) == True:
6
            count = count+1
7
        else:
8
            count = count
9
print(count, "of my favorite songs have long titles!")
```

Style Error 1:
Line number:

Style Error 2:
Line number:

Style Error 3:
Line number:

Style Error 4:
Line number:

---

23. Read the following code and traceback. Then, in the space below, explain the cause

of the error

Code Snippet Traceback

```
 1 def foo():
 2 l = list()
 3 odd_numbers = [1, 3, 5, 7, 9]
 4 even_numbers = [2, 4, 6, 8, 10]
 5 l.append(odd_numbers)
 6 l.append(even_numbers)
 7 baz(l, None)
 8 bar(l)
 9
10
11 def bar(l):
12 d = dict()
13 d["odd_numbers"] = l[0]
14 d["even_numbers"] = l[1]
15 d["numbers"] = l
16 baz(l, d)
17
18
19 def baz(l, d):
20 s = set()
21 if d is not None:
22 s.add(tuple(d.keys()))
23 s.add(d["numbers"][1])
24 else:
25 s.add(l[0][0])
26
27
28 def main():
29 foo()
30
31
32 if __name__ == "__main__":
33 print("Welcome to the Final!")
34 main()
```

```
Welcome to the CSE 160 Final!
Traceback (most recent call last):
File ".../error.py", line 34, in
<module>
    main()
  File ".../error.py", line 29, in main
    foo()
  File ".../error.py", line 8, in foo
    bar(l)
  File ".../error.py", line 16, in bar
    baz(l, d)
  File ".../error.py", line 23, in baz
    s.add(d["numbers"][1])
TypeError: unhashable type: 'list'
```

---

24. For parts 1 and 2 of this question, consider the following code:

```
moe = 5.5

def eeny(num):
      if (num - 5) > 0:
            return meeny(moe)
      else:
            return meeny(num) * 20

def meeny(moe):
      if moe % 2 == 0:
            return miny(2, 6) * 2
      else:
```

```
            return miny(moe, 3) * 20

def miny(num, moe):
      if moe * num > 12:
            return 12
      else:
            return moe * num
```

Part 1:
For each of the following, what will print out?

Code Output

```
print(eeny(10))


print(eeny(4))


print(eeny(-1))
```

Part 2:
Provide another example call to `eeny()` that would print out the number `480`. **Note**: You may not use any of the calls to eeny from the previous question. E.g., you cannot write `print(eeny(10))`, but you could write `print(eeny(9))`.

---

25. A researcher has written the function below to be able to find how many times a specific string repeats in a list:

```
def CountTotals(list, i):
      count = 0
      for current_string in list:
            if current_string == i:
                  count += 1
      return count
```

What three main style issues should the researcher fix, and give an example of what should be changed to fix the issues.

Style

Issue

Example

Fix

---

26. We've written the following function to help us find the longest name in a text file listing the names of some CSE160 staff members:

```
def longest_name(data):
      """
      Returns the longest name in the given parameter file. If there
      is a tie for longest name, returns the first name with the
      longest length. """
      myfile = open(data)

      longest_name = None

      for persons in myfile:
            for person in persons.split():
                  if longest_name is None or len(persons) >
            len(longest_name):
                        longest_name = person

      myfile.close()
      return longest_name
```

When we run

```
      print(longest_name("data.txt"))
```

… where the file named "`data.txt`" has the following contents

```
      Fitz Zoe
      Paolo Mark Annalisa
      Sneh Amanda Erica
```

… the code prints "Erica", which has 5 letters, while the longest name we are looking for is "Annalisa", which has 8 letters. Take a look at the function's implementation and circle which line of code is causing the bug. Then, below, explain how we can fix the bug so that our code produces the correct output.

---

27. Consider the following code:

```
def mystery(lst):
     result = [i ** 2 if i % 2 == 1 else i // 2 for i in lst]
     return result
```

For the below problems, you will be asked to provide a call to the `mystery()` function. One example call of this function is: `mystery([0, 0])`, where the parameters provided inside of the function can differ.

a. Provide an example call to the `mystery()` function that would return the list `[7, 49, 0, 1]`.

b. Provide an example call to the `mystery()` function that would return the list `[25, 9, 16, 81]`.

---

28. Consider the following class definition:

```
from operator import itemgetter
class MusicShow:
     def __init__(self, show_name):
          self.show_name = show_name
          self.performers = []
     def add_artist(self, artist, members):
          self.performers.append((artist, members))
          print(self.show_name, "added", artist)
     def print_top_list(self, num=3):
          top_list = sorted(self.performers, key=itemgetter(0),
          reverse=True)
          top_list = sorted(top_list, key=itemgetter(1))
          print(top_list[:num])
```

For the below code snippet, write out the expected printed output.

```
music_bank = MusicShow("Music Bank")
music_bank.add_artist("wayv", 6)
music_bank.add_artist("enhypen", 7)
music_bank.add_artist("ive", 6)
music_bank.add_artist("ateez", 8)
music_bank.print_top_list()
```

For the below code snippet, write out the expected printed output.

```
the_show = MusicShow("The Show")
artists = ["iu", "red velvet", "sejeong", "le sserafim", "aespa"]
members = [1, 5, 1, 5, 4]
for i in range(len(artists)):
the_show.add_artist(artists[i], members[i])
the_show.print_top_list(4)
```

---

29. A student wrote a function that calculates if the average age in `age_list` is greater than the given parameter `age`. The function returns True if the average age is greater than `age`, False otherwise.

However, the student also made some style issues. Identify four style issues that the student should fix and the corresponding change to fix the issue.

```
def checkAverageAge(age, age_list):
      sum = 0
      for curr_age in age_list:
            sum = sum + curr_age
      if sum/len(age_list) > age:
            return True
      else:
            return False
```

Style Issue	Solution

---

30. A student is writing tests for the HW4 function `euclidean_distance`. For a refresher, the docstring and code is below:

```
""" Calculate the Euclidean distance between two data points.
    Arguments:
```

```
        point1: a non-empty list of floats representing a data point
        point2: a non-empty list of floats representing a data point
    Returns: the Euclidean distance between the two data points

    Example:
    >>> euclidean_distance([1.1, 1, 1, 0.5], [4, 3.14, 2, 1])
    3.7735394525564456
"""
total = 0
for i in range(len(point1)):
     total += (point1[i] - point2[i]) ** 2
return math.sqrt(total)
```

The tests that they have written contain errors or have made some incorrect assumptions. Find what they are and write a new test to fix the issue. (For the sake of hand-writing time, you may abbreviate "`euclidean_distance`" as "`euc_dist`".)

Note: Each test has a different issue. In order to get full credit for this question, you must list different issues for each test; writing the same issue for more than one test will not earn full points.

```
# Test 1
assert euclidean_distance(4, 5) == 1
```

Test 1's Issue	**New Test 1**

```
# Test 2
assert euclidean_distance([1, 1], [1, 0]) == 2
```

Test 2's Issue	**New Test 2**

```
# Test 3
assert euclidean_distance([1, 0, 1], [0, 1]) == 3
```

Test 3's Issue	New Test 3

```
# Test 4
assert euclidean_distance([1.1, 1, 1, .5], [4, 3.14, 2, 1]) == 3.7735
```

Test 4's Issue	New Test 4

---

31. While working on HW3, you receive this error message:

```
Traceback (most recent call last):
  File "...homework3\blur_image.py", line 266, in <module>
    test_blur()
  File "...homework3\blur_image.py", line 261, in test_blur
    assert blur(test_grid) == blurgrid
  File "...homework3\blur_image.py", line 243, in blur
    blurred_pixel = average_of_surrounding(pixel_grid, i, j)
  File "...homework3\blur_image.py", line 203, in
average_of_surrounding
    pixel_sum += get_pixel_at(pixel_grid, row, col)
  File "...homework3\blur_image.py", line 142, in get_pixel_at
    return pixel_grid[i][j]
IndexError: list index out of range
```

1. Which of the following HW3 functions should you look at first to solve this error? Circle ***one***.

    a. `test_blur()`

b. `average_of_surrounding(pixel_grid, i, j)`

c. `get_pixel_at(pixel_grid, row, col)`

2. In your own words, what does `IndexError` mean? (in other words, how might this error have occurred?)

3. In two lines or less, write code that would produce the same error.

---

32. While working on HW5, you receive this error message.

```
Traceback (most recent call last):
  File "...homework5\social_network_tests.py", line 132, in <module>
    main_test()
  File "...homework5\social_network_tests.py", line 85, in
test_num_map_to_sorted_list
    assert_equals(['c', 'a', 'd', 'e', 'b'],
  File "...homework5\utils.py", line 59, in assert_equals
    assert check_approx_equals(expected, received), \
AssertionError: Failed: Expected ['c', 'a', 'd', 'e', 'b'],
                       but received ['b', 'e', 'd', 'a', 'c']
```

1. Which of the following HW5 functions should you look at first to fix the error? Circle ***one***.

   a. `num_common_friends_map(graph, user)`

   b. `num_map_to_sorted_list(map_with_number_vals)`

   c. `rec_number_common_friends(graph, user)`

   d. `main_test()`

   e. `assert_equals(expected, received)`

   f. `check_approx_equals(expected, received)`

   g. `recommend_by_influence(graph, user)`

2. What does `AssertionError` mean?

3. Give one plausible reason why this error might have occurred (Hint: what is the difference between the expected and received output?)

Code Writing

33. For this problem you should write code in good style (as if you were submitting it for a homework assignment) and you should use functions defined in earlier parts of the problem if applicable.

You have a data structure which is a list of dictionaries as follows:

```
cities = [
    {'Name': 'Vancouver', 'State': 'WA', 'Population': 161791},
    {'Name': 'Salem', 'State': 'OR', 'Population': 154637},
    {'Name': 'Seattle', 'State': 'WA', 'Population': 608660},
    {'Name': 'Spokane', 'State': 'WA', 'Population': 208916},
    {'Name': 'Portland', 'State': 'OR', 'Population': 583776}
    {'Name': 'Bellingham', 'State': 'WA', 'Population': 80885},
]
```

a. Complete the function `max_in_state` to return the city (as a dictionary) with the highest population in a given state. If `cities` contained only the dictionaries above, a call to `max_in_state(cities, 'WA')` would return:

```
{'Name': 'Seattle', 'State': 'WA', 'Population': 608660}
```

```
def max_in_state(city_list, state):
    '''
    Return the dictionary for the city in city_list with the largest
    population for the given state. Return None if there are no
cities
    for the given state in city_list.
    '''
```

b. Complete the following function, state_population, to return the total population for a given state. (Assume that all census locations for a state are included in the cities list).

If cities contained only the following dictionaries:

```
cities = [
    {'Name': 'Seattle', 'State': 'WA', 'Population': 50},
    {'Name': 'Portland', 'State': 'OR', 'Population': 10},
    {'Name': 'Spokane', 'State': 'WA', 'Population': 20},
    {'Name': 'Vancouver', 'State': 'WA', 'Population': 7},
]
```

a call to `state_population(cities, 'WA')` would return: `77`

```
def state_population(city_list, state):
    '''
    Return a number representing the total population of the given
state
    based on the list of cities provided. Return None if there are no
cities
    for the given state in city_list.
    '''
```

c. Complete the following function to return a dictionary mapping state to the total population for that state. (You should assume that all census locations for a state are included in the cities list.)

If cities contained only the dictionaries shown in Part b, a call to `all_state_populations(cities)` would return: `{'WA': 77, 'OR': 10}`

```
def all_state_populations(city_list):
    '''
    Returns a dictionary mapping state to total population  for each
state in
    city_list. Returns an empty dictionary if city_list is empty.
    '''
```

---

34. You are given the following class definition:

```
class Car:
    def __init__(self, gas_tank_size, miles_per_gallon):
        '''
        gas_tank_size and gas are in gallons
        '''
        self.gas = 0.0
        self.tank_size = gas_tank_size
        self.mpg = miles_per_gallon

    def refill_tank(self):
        '''
        Fills gas tank up to the tank size. tank_size and gas are in
gallons.
        '''
        self.gas = self.tank_size
```

a. Fill in the code for the function below that is also a part of the class Car.

```
    def drive(self, miles):
        '''
        Drives the car the provided number of miles. Will deduct the
        required gas from the tank of the car. If there is not enough
gas
        just print "Not enough gas!" and will not drive or deduct the
gas.
        '''
        # Your code here
```

b. Add code below to use the method that you wrote to drive nicks_car 16 miles.

```
nicks_car = Car(15, 30)
nicks_car.refill_tank()
# Your code here
```

c. List one advantage of using a class to represent a Car

---

35. Given a list of lists of integers, write the function `index_of_max_unique` below to return the index indicating which sub-list has the most unique values. For example:

`index_of_max_unique([[1, 3, 3], [12, 4, 12, 7, 4, 4], [41, 2, 4, 7, 1, 12]])` would return 2 since the sub-list at index 2 has the most unique values in it (6 unique values).

`index_of_max_unique([[4, 5], [12]])` would return 0 since the sub-list at index 0 has the most unique values in it (2 unique values).

You can assume that neither the `list_of_lists` nor any of its sub-lists will be empty. If there is a tie for the max number of unique values between two sub-lists, return the index of the first sub-list encountered (when reading left to right) that has the most unique values.

```
def index_of_max_unique(list_of_lists):
     # Your code here
```

---

36. Write a function called union_sets that takes two lists of sets of integers as arguments. Assume the two lists are of equal length. The function should return a list containing the unions of the two sets in the corresponding positions of each list. For example:

```
list_a = [{1}, {3, 4, 5}, {1, 2}]
```

```
list_b = [{3}, {3, 5, 6}, {2}]
print union_sets(list_a, list_b)
```

Would print: `[set([1, 3]), set([3, 5, 4, 6]), set([1, 2])]`

```
def union_sets(lst_one, lst_two):
    # Assumes lst_one and lst_two contain at least one set each.
    # Your code here
```

37.a. Write a function called `read_expenses(filename)` that takes the name of a file as a parameter. You can assume that the given file contains lines of the form: `store_name amount_spent` where `store_name` is a string (that contains no spaces or punctuation) and `amount_spent` is a float. The two values are separated by a single space. Here are the contents of a sample input file:

```
varsity_theater 10.56
shultzys 15.34
solstice_cafe 5.01
shultzys 4.50
solstice_cafe 6.0
```

Your function should read in the given file and return a dictionary mapping each `store_name` to the total amount of money you have spent there. You may assume the file name provided is valid and that the file is formatted as described and includes at least one `store_name amount_spent` pair.

Calling read_expenses on the example file above returns this dictionary:
`{'varsity_theater': 10.56, 'shultzys': 19.84, 'solstice_cafe': 11.01}`

```
def read_expenses(filename):
    # Your code here
```

b. Describe 3 things about your approach to testing this function.

c. Write code in the main function that will:

- call the `read_expenses` function written in part (a) to read a file called `"jan.txt"`.
- print out the expenses sorted from the store where you spent the most to the store where you spent the least (if you spent the same amount at more than one store, sort alphabetically from a to z by store name). You should print the results in EXACTLY the following format. For the sample input file shown in part (a) the output would be:

```
Spent 19.84 at shultzys
```

```
Spent 11.01 at solstice_cafe
Spent 10.56 at varsity_theater

from operator import itemgetter
def main():
    # Your code here
```

---

38. You are given the following class definition:

```
class MovieRatings:
    def __init__(self, user_name):
        '''
        user_name: a string representing the name of the person these
                   movie ratings belong to
        '''
        self.name = user_name
        self.scores = {}

    def rate(self, movie_name, rating):
        '''
        movie_name: a string representing a movie
        rating: a float representing this user's rating of the movie
        '''
        self.scores[movie_name] = rating
```

a. Write the code for the method below that is also a part of the class `MovieRatings`:

```
    def get_highest_rating(self):
        '''
        Returns a float representing the highest rating of all the
movies
        this user has rated. Returns 0 if the user has not yet rated
any
        movies.
        '''
        # Your code here
```

b. Write code in the main function, using methods from the MovieRatings class, to:

- add a rating of 9.5 for “Moonlight” to marys_ratings.
- Print Mary Jones’ highest rating

This code is outside of the class MovieRatings.

```
def main():
    marys_ratings = MovieRatings("Mary Jones")
    # Your code here:
    marys_ratings.rate("Moonlight", 9.5)
    print marys_ratings.get_highest_rating()
```

c. Describe a change to the `MovieRatings` class that would NOT cause a client of the `MovieRatings` class to have to modify its code.

d. Describe a change to the `MovieRatings` class that might cause a client of the `MovieRatings` class to have to modify its code.

---

39. Write a function called favorite_food(filename) that takes the name of a file as a string parameter. Each line of the file is in the form:

```
TA_name food_item1 food_item2 ...
```

The TA_name and each food_item are strings (containing no spaces or punctuation). The number of food items given for each TA is variable (some TAs only like one food item, while some TAs like many food items). Additionally, some TAs decided to add more items they liked later to the file, so there can be multiple lines in the file with the same TA and additional favorite food items. Note, food items are unique and never repeated.

Here are the contents of a sample input file, `TAFavoriteFood.txt`:

```
Sneh Pizza Pasta
Wen Popcorn
Analisa Sushi Cake Cookies Coffee
Max Teriyaki
Sneh Gyro
```

Your function should read in the given file and return a dictionary mapping each TA to a list of their favorite food(s). You may assume the file name provided is valid and that the file is formatted as described and includes at least one valid line.

Expected output when calling **`favorite_food("TAFAvoriteFood.txt"):`**

```
{"Sneh": ["Pizza", "Pasta", "Gyro"], "Wen": ["Popcorn"],
 "Annalisa": ["Sushi", "Cake", "Cookies", "Coffee"],
 "Max": ["Teriyaki"]}
```

```
def favorite_food(filename):
    # Write your code here
```

---

40. **Part A:** Write a function called **`election_results`** that takes a single parameter (**`vote_counts`**, a dictionary) that maps candidate names to how many votes they received. The function should then return a list that represents the individual votes. For example if vote_counts is defined as

```
vote_counts {"john" : 4, "johnny" : 3, "jackie" : 2,
```

`"jamie" : 4}` then a function call of

```
election_results(vote_counts)
```

would return the list

```
["john", "john", "john", "john", "johnny", "johnny",
 "johnny", "jackie", "jackie", "jamie", "jamie", "jamie",
 "jamie"]
```

The order of the returned votes does not matter.

```
def election_results(vote_counts):
    # Write your code here
```

Part B: Write **two** distinct tests, in the form `assert <condition>` (e.g., `assert 1 != 2`), that will validate that the function you wrote in Part A works as expected. Each test should test a different case than the other. Then, below those tests, write one to two sentences that describe why those test cases are different from each other.

---

41. Write a function `divisible` that, given two lists of integers `products` and `factors`, *returns* the number of ints in `products` divisible by any one of the factors.

Examples:
`divisible ([36, 45, 7], [3, 11])` returns `2`
Explanation: 36 is divisible by 3 and 45 is divisible by 3
`divisible ([36, 45, 7], [3, 5])` returns `2`
Explanation: 36 is divisible by 3 and 45 is divisible by 3 or 5, but is still only

counted once `divisible ([36, 45, 48], [2, 3, 6])` returns `3`
Explanation: 36 is divisible by 2,3, and 6; 45 is divisible by 3; 48 is divisible by 2, 3 and 6 `divisible ([36, 45, 48], [7, 11])` returns `0`
Explanation: None of the numbers are divisible by 7 or 11

Even if a product is divisible by multiple factors, it should only be counted once (see examples 2 and 3).

```
def divisible(products, factors):
# Write your code below this line
```

---

42. Write 3 test cases for the following function that test 3 different cases. Please specify what the cases are testing (as in, why they are different from the other two tests) as well as the value of the input variables. Remember to clearly define the input you will use to call the function!

```
def find_common(TA_food, TA1, TA2):
"""
Parameters:
TA_food: dictionary where the keys are TAs, and the values are a
set of the TA's favorite food
TA1: a string representing a TA in TA_food
TA2: a string representing another TA in TA_food

Returns: a set of the food items that both TAs like. If the two TAs
do not have any common favorite foods, return an empty set.

Example:
>>> TA_food = {'Sneh': {'pizza', 'pasta'}, 'Zoe': {'pizza', 'sushi'},
'Sierrah': {'Apple pie', 'chicken teriyaki'}}
>>> find_common(TA_food, 'Sneh', 'Zoe')
{'sushi', 'pizza'}
"""
```

---

43. For parts 1 and 2 of this question, you will write and use a class designed to store information about certain kinds of songs.

Part 1: Complete the missing two class functions below. The descriptions for each function are provided in the docstrings.

```
class EurovisionSong:
     def __init__(self, name, length, country):
```

```
        """
        length is given in seconds (int)
        name and country are given as strings
        """
        # Fill in the initialization for the three provided
        attributes:

    def get_length(self):
        """
        Returns the length of the song as a tuple (hours,
        minutes, seconds).
        For example:
        >>> s = EurovisionSong("Heart of Steel", 154,
        "Ukraine") >>> s.get_length()
        (0, 2, 34)
        """
        # Write your implementation here:
```

Part 2: Write a function that takes a list of EurovisionSong instances and returns a dictionary mapping country names (keys) to the name of the longest song for that country.

For example:

```
songs = [EurovisionSong("Ein bisschen Frieden", 197,
        "Germany"), EurovisionSong("Fairytale", 183,
        "Norway"),
        EurovisionSong("Hard Rock Hallelujah", 204,
        "Finland"), EurovisionSong("Waterloo", 169,
        "Sweden"),
        EurovisionSong("Satellite", 173, "Germany"),
        EurovisionSong("La det swinge", 180, "Norway")]
map_songs(songs)
```

would return:

```
{"Germany": "Ein bisschen Frieden"
"Norway": "Fairytale",
"Finland": "Hard Rock Hallelujah"
"Sweden": "Waterloo"}


def map_songs(songs):
    # Write your implementation here:
```

---

44. Write a function set combination that takes in a list of lists and returns a set where the

elements appear at least once in an inner list.

For example:
- `set_combination([[]])` should return `set()` (an empty set).
- `set_combination([[1], [2], [3]])` should return `{1, 2, 3}`.
- `set_combination([[2, 4], [4, 6], [6, 8]])` should return `{2, 4, 6, 8}`.

```
def set_combination(list_of_lists):
    # Your code goes here
```

---

45. Suppose you are given a file named `"gpa.csv"` where there are three columns in the CSV file: `student_name, class_name,` and `gpa`.

You should write code to print out a dictionary where the keys are each `student_name` in the csv file and the values are a list of tuples, where each tuple consists of two elements (`class_name, gpa`), where `class_name` is a string and `gpa` is a float. If a student appears multiple times in the file, the dictionary's list value associated with the student will contain multiple tuples of class names with their respective GPAs.

For example, given an example `"gpa.csv"` file that contains the following content:

```
student_name,class_name,gpa
Lucas,CSE 160,3.0
Sheamin,INFO 340,3.8
Suh Young,CSE 160,3.8
Suh Young,CLAS 205,4.0
```

You should print out the dictionary:

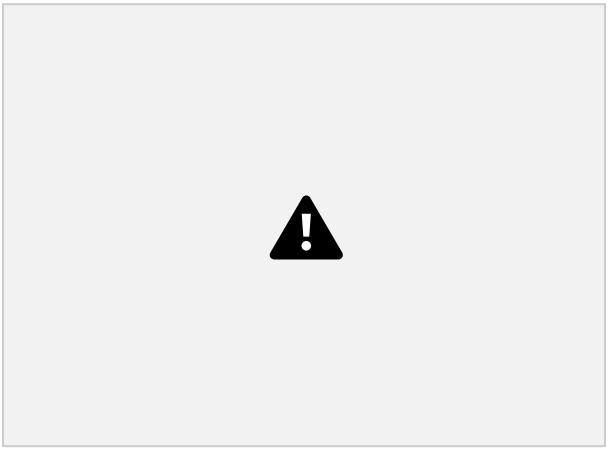
```
{"Lucas": [("CSE 160", 3.0)], "Sheamin": [("INFO 340", 3.8)], "Suh
Young": [("CSE 160", 3.8), ("CLAS 205", 4.0)]}
```

```
import csv # You can use this library in your code
# Your code goes here
```

---

46. Write a function max edges that takes in a graph g and returns the value of the node that has the maximum edges. You can assume that only one node has the maximum amount of edges. You should not use the max function to solve this problem.

For example, given this graph g:



You should return "E", because that node has the most amount of edges in the graph (four edges) compared to other nodes.

```
import math
import networkx as nx


def max_edges(g):
# Your code goes here
```

---

47. Write three test cases that test different behaviors using assert statements for each of the following functions given its docstring:

a. `get_max_key(my_dict)`: Given a dictionary `my_dict` where keys are strings and values are floats, returns the key that has the corresponding maximum value. You can assume `my_dict` will only contain one key with the maximum float value. Returns None if `my_dict` is empty.

b. `count_chars(lst, char)`: Given a list of strings `lst` and a single character string `char`, return a dictionary where the keys are each individual element in `lst` and the value is the respective count of the character char for each key (case-sensitive). Return an empty dictionary if `lst` is empty.

---

48. Suppose you are given a CSV file with the following headers:

```
section, enrollment, median_score, avg_score
```

Write a function that takes the filename as a parameter, reads the file, and then returns a dictionary where the keys are the class section and the values are a dictionary of that section's statistics. You may assume that all numbers in the data are integers and that the **`csv`** module has been imported. For example:

```
class_stats("stats.csv")
```

where the `stats.csv` file contains

```
section,enrollment,median_score,avg_score
AB,15,72,77
AC,8,82,83
```

should return the dictionary

```
{
 "AB": { "enrollment": 15, "median_score": 72, "avg_score": 72
},
 "AC": { "enrollment": 8, "median_score": 82, "avg_score": 83 }
}
```

Write your solution below this function header:

```
def class_stats(filename):
```

---

49. Color images are represented as a grid of pixels (like the grayscale images from HW3), but each pixel actually has three values. These values correspond to the amount of red, green, and blue in that location of the image. For instance, a 4-pixel by 4-pixel image could be represented as follows:

```
image = [
          [[0,  0,  0],  [1,  1,  1],  [2,  2,  2],  [3,  3,  3]],
          [[4,  4,  4],  [5,  5,  5],  [6,  6,  6],  [7,  7,  7]],
          [[8,  8,  8],  [9,  9,  9],  [10, 10, 10], [11, 11, 11]],
          [[12, 12, 12], [13, 13, 13], [14, 14, 14], [15, 15, 15]]
        ]
```

The first element in the innermost lists represents the amount of red, the second represents the amount of green, and the third represents the amount of blue.

Write a function that accepts the row and column of a pixel and returns a dictionary of the color components. The keys in the returned dictionary should be the colors red, blue, and green and the values are the respective quantities of the color.

For example:
`pixel_color_info(image, 1, 3)` should return `{'red': 7, 'blue': 7, 'green': 7}`
`pixel_color_info(image, 2, 2)` should return `{'red': 10, 'blue': 10, 'green': 10}`

`def pixel_color_info(image, row, col):`

---

50. Consider the following class for all parts of question 7:

```
class RNA():
    '''
    A class that represents an RNA molecule.
    The RNA class stores its name and sequence on construction.
    Member functions compute useful data about an RNA instance.
    '''

    def __init__(self, name, sequence):
        '''
        Creates an RNA sequence with a given name and sequence, both strings.
        '''
        self.name = name
        self.sequence = sequence

    def __len__(self):
        '''
        Returns the number of bases in the RNA sequence

        Example:
        >>> r = RNA("test", "ACGUACGU")
        >>> len(r)
        8
        '''
        return len(self.sequence)

    def __getitem__(self, index):
        '''
        Returns the base nucleotide at the given index. This enables support
```

```
        for "subscripting"--i.e., list-like brackets notation.

        Examples:
        >>> r = RNA("test", "ACGUACGU")
        >>> r[4]
        'A'
        >>> r['A']
        Traceback (most recent call last):
            ...
        AssertionError: Index must be an integer
        '''
        assert type(item) is int, "Index must be an integer"
        return self.sequence[index]

    def aa_to_codon(self):
        '''
        Returns dictionary mapping of amino acids to a list of their possible
        RNA sequence codons.
        '''
        aa_map = {
            "Leu": ["UUA", "UUG", "CUU", "CUC", "CUA", "CUG"],
            "Val": ["GUU", "GUC", "GUA", "GUG"],
            "Met": ["AUG"],
            "Pro": ["CCU", "CCC", "CCA", "CCG"],
            "Ala": ["GCU", "GCC", "GCA", "GCG"],
            "Trp": ["UGG"],
            "Arg": ["CGU", "CGC", "CGA", "CGG", "AGA", "AGG"],
            "Gly": ["GGU", "GGC", "GGA", "GGG"],
        }
        return aa_map
```

<End of RNA class>

The following questions (Question 7 Parts A through D) ask you to write the bodies of some missing function. Pay close attention to what each method's documentation expects and what methods the class already provides (above).

Part A) Write the body of the following RNA class method, adhering to the description and behavior defined in its docstring.

```
def get_percent_GC_content(self):
    '''
    Returns percent of 'G' and 'C' bases in the sequence.

    Examples:
    >>> r1 = RNA("test1", "UAGUAGGUUG")
    >>> r1.get_percent_GC_content()
    40.0
```

```
    >>> r2 = RNA("test2", "UAAUAUUUUG")
    >>> r2.get_percent_GC_content()
    10.0
    '''
```

Part B) Write the body of the following RNA class method, adhering to the description and behavior defined in its docstring.

```
def get_reverse_complement(self):
    '''
    Returns the reverse complement of the RNA sequence. The compliment is
    calculated as swapping A <-> U and C <-> G. The complement is then
    reversed.

    Examples:
    >>> r1 = RNA("test1", "AUGCAGC")
    >>> r1.get_reverse_complement()
    'GCUGCAU'
    >>> r2 = RNA("test2", "GCUGCAU")
    >>> r2.get_reverse_complement()
    'AUGCAGC'
    '''
```

Part C) Write the body of the following RNA class method, adhering to the description and behavior defined in its docstring.

```
def translate_rna(self):
    '''
    Translates an RNA sequence to the correct amino acid chain. The RNA
    sequence should be read in consecutive, non-overlapping 3-base codons,
    starting with index 0.

    Returns a list containing the three letter amino acid codes.

    Examples:
    >>> r1 = RNA("test1", "AUGGCCA")
    >>> r1.translate_rna()
    ['Met', 'Ala']
    >>> r2 = RNA("test2", "UGGGAUGAGGUAGUAGGUU")
    >>> r2.translate_rna()
    ['Trp', 'Val', 'Val', 'Gly']
    '''
```

7 Part D) Write the body of the following RNA class method, adhering to the description and behavior defined in its docstring.

```
def distance(self, other):
    '''
    Calculates a simplified distance between two RNA sequences. For this
    problem, we define distance as the absolute value of the difference in
    sequence lengths plus the number of mismatched nucleotides. (In real
    RNA computations, the distance calculation is much more complex.)

    Examples:
    >>> r1 = RNA("seq1", "ACGT")
    >>> r2 = RNA("seq2", "TAGC")
    >>> r1.distance(r2)
    3
    >>> r3 = RNA("seq3", "ACG")
    >>> r4 = RNA("seq4", "ACGTA")
    >>> r3.distance(r4)
    2
    >>> r4 = RNA("seq3", "CCCAAA")
    >>> r5 = RNA("seq4", "AAA")
    >>> r4.distance(r5)
    6
    '''
```