Full Name: **Answer Key** _____

Email Address (UW Net ID): _____ @uw.edu

Section: _____

-

CSE 160 Winter 2024 - Final Exam

Instructions:

- You have **110 minutes** to complete this exam.
- The exam is **closed book**, including no calculators, computers, phones, watches or other electronics.
- You are allowed a single sheet of notes for yourself.
- We also provide a syntax reference sheet.
- Turn in *all sheets* of this exam, together and in the same order when you are finished.
- When time has been called, you must put down your pencil and stop writing.
    - **Points will be deducted if you are still writing after time has been called.**
- You may only use parts and features of Python that have been covered in class up to this point.
- You may ask questions by raising your hand, and a TA will come over to you.

Good luck!

Question	**Topic(s)**
Question 1	Expressions, data types
Question 2	Lists, File I/O
Question 3	Nested lists, dictionaries
Question 4	Testing
Question 5	Tracebacks, errors
Question 6	Tracebacks, errors
Question 7	Classes, methods/functions

1) 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`	**str**	**"HelloWorldHelloWorld"**
`x = print("not an integer?")`	**NoneType**	**None**
`x = "one" + "one" == "two"`	**bool**	**False**
`x = 3 * 3 / 3`	**float**	**3.0**
`x = sorted("python")`	**list**	**['n', 'o', 'h', 't', 'y', 'p']**
`x = "python"[::-1]`	**str**	**"nothyp"**

`a = {'a', 'b', 'c'}`
`z = {'x', 'y', 'z'}`
`x = a & z`
	set	**{} (empty set)**

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

2) 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):
    file = open(filename)
    reader = csv.DictReader(file)
    result = {}
    for row in reader:
        section = row['section']
        result[section] = {}
        for key in row:
            if key != 'section':
                result[section][key] = int(row[key])

    file.close()
    return result
```

3) 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 blue, and the third represents the amount of green.

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):
    return {
        'red': image[row][col][0],
        'green': image[row][col][1],
        'blue': image[row][col][2]
    }
```

4) 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**
Arguments are not lists	**`assert euclidean_distance([4], [5]) == 1`**

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

Test 2's Issue	**New Test 2**
Output number is not correct	**`assert euclidean_distance([1,1], [1,0]) == 1`**

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

Test 3's Issue	New Test 3
List lengths are mismatched	assert euclidean_distance([1, 0, 1], [0, 1, 0]) == 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
Need to use math.isclose for floats	assert math.isclose(euclidean_distance([1.1, 1, 1, .5], [4, 3.14, 2, 1]), 3.7735)

5) 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. What does `IndexError` mean? (in other words, how might this error have occurred?)

It means that the list (in this case pixel_grid) does not have the index requested (in this case i or j)

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

```
[1, 2, 3][4]
```

6) 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?

It means that the condition given to an assert statement evaluated to False.

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

The order of the elements returned from num_map_to_sorted_list is incorrect. Most likely forgot to reverse the resulting sort.

7) 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]
```

<RNA class continues on next page>

<RNA class continued from previous page>

```
    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).

7 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
    '''
    count = 0
    total_bases = len(self)
    for base in self.sequence:
        if base == 'G' or base == 'C':
            count += 1
    gc_content = (count / total_bases) * 100
    return gc_content
```

7 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'
    '''
    complement_dict = {'A': 'U', 'U': 'A', 'C': 'G', 'G': 'C'}
    complement = ''
    for base in self.sequence:
        complement_base = complement_dict[base]
        complement += complement_base
    reverse_complement = complement[::-1]
    return reverse_complement
```

7 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']
    '''
    n = len(self.sequence)-1
    aa_chain = []
    for i in range(0, n, 3):
        # set codon variable to slice three nucleotides per codon
        codon = self.sequence[i:i+3]
        for aa, codons in self.aa_to_codon().items():
            if codon in codons:
                aa_chain.append(aa)
    return aa_chain
```

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
    '''
    distance = abs(len(self) - len(other))
    min_len = min(len(self), len(other))
    for i in range(min_len):
        if self[i] != other[i]:
            distance += 1
    return distance
```

Extra Credit (1pt):

Choose one (or both, but you will only receive a maximum of 1 point of extra credit):

1) Write a poem, limerick, haiku, acrostic, epic, rhyming couplet, or any other literary form about your experience in CSE 160 this quarter.

2) What was your favorite assignment in this class? Why is it your favorite?