HW1 – Kevin Lopez Sepulveda

- **Submission date: 01/31/2025 (11:59 PM)**
- Revise the topics listed on Slide 9 in the Class 1 slide deck *thoroughly* before attempting the assignment.
- You can submit ***well commented*** code in either `.py` or `.ipynb` formats.
- Submitting a PDF explaining your code is *highly* encouraged.

1. **List and dictionary comprehension.**
   a. Given a list `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, write a list comprehension that generates the squares of all elements of `numbers`.
      i Check python file
   b. What is title casing? Given a list `fruits = ['green apple', 'banana', 'cherry', 'date']`, write a list comprehension that title cases each element of the list `fruits`. Your output should look like: `['Green Apple', 'Banana', 'Cherry', 'Date']`.
      i Check python file and used title casing in list comprehension

   c. Using dictionary comprehension, create a dictionary, `nested_dictionary`, where the keys are numbers from 1 to 3, and the values are dictionaries where the keys are the first three letters of the alphabet, and the values are the product of the outer and inner keys. Assume `'a'` corresponds to `1`, `'b'` corresponds to `2`, …, and `'z'` corresponds to `26`. Your output should look like:
      ```
      {1: {'a': 1, 'b': 2, 'c': 3}, 2: {'a': 2, 'b': 4, 'c': 6}, 3: {'a': 3, 'b': 6, 'c': 9}}
      ```
      Hint: The built-in function `ord()` returns the Unicode number for one-character strings.
      i Check python file

2. **Anonymous functions and applications.** (some information about the `int()` function)
   a. Write a function that uses `map()` to convert a list of binary strings into their corresponding decimal values. Each binary string represents a non-negative integer.
      Example input: `binaries = ['1010', '1111', '0001', '100000']`
      Output: `[10, 15, 1, 32]`
      i Check python file

   b. Use `filter()` to create a list of strings from the given list that are palindromes. A palindrome is a string that reads the same forward and backward.
      Example input: `words = ['level', 'world', 'deified', 'python', 'radar']`
      Output: `['level', 'deified', 'radar']`

c. Write a single line of Python code using `map()` and `filter()` to find the squares of even numbers from the given list of integers.
Input: `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
Output: `[4, 16, 36, 64, 100]`

d. Given a list of integers, use `filter()` to create a new list that contains only those integers that are prime numbers.

3. **File I/O, functions and built-in functions.** The file "`gettysburg.txt`" contains the text of President Lincoln's famous 1863 address.
   a. Open the file for reading. Assume your code and the file are in the same folder.
   b. Using the `split()` and `extend()` methods define a function `speech_to_list()` that takes the contents of the file and outputs a list containing all the words in the file. Store this in a list called `speech_list`.
   Your output should look like this:


```
print(speech_list)
```

```
['Four', 'score', 'and', 'seven', 'years', 'ago', 'our', 'fathers', 'brought', 'forth', 'on', 'this', 'continent', 'a', 'new', 'nation,', 'conceived',
'in', 'Liberty,', 'and', 'dedicated', 'to', 'the', 'proposition', 'that', 'all', 'men', 'are', 'created', 'equal.', 'Now', 'we', 'are', 'engaged', 'in',
'a', 'great', 'civil', 'war,', 'testing', 'whether', 'that', 'nation,', 'or', 'any', 'nation,', 'so', 'conceived', 'and', 'so', 'dedicated,', 'can', 'lo
ng', 'endure.', 'We', 'are', 'met', 'on', 'a', 'great', 'battle-field', 'of', 'that', 'war.', 'We', 'have', 'come', 'to', 'dedicate', 'a', 'portion', 'o
f', 'that', 'field,', 'as', 'a', 'final', 'resting', 'place', 'for', 'those', 'who', 'here', 'gave', 'their', 'lives', 'that', 'that', 'nation', 'migh
t', 'live.', 'It', 'is', 'altogether', 'fitting', 'and', 'proper', 'that', 'we', 'should', 'do', 'this.', 'But,', 'in', 'a', 'larger', 'sense,', 'we',
'can', 'not', 'dedicate', '--', 'we', 'can', 'not', 'consecrate', '--', 'we', 'can', 'not', 'hallow', '--', 'this', 'ground.', 'The', 'brave', 'men,',
'living', 'and', 'dead,', 'who', 'struggled', 'here,', 'have', 'consecrated', 'it,', 'far', 'above', 'our', 'poor', 'power', 'to', 'add', 'or', 'detrac
t.', 'The', 'world', 'will', 'little', 'note,', 'nor', 'long', 'remember', 'what', 'we', 'say', 'here,', 'but', 'it', 'can', 'never', 'forget', 'what',
'they', 'did', 'here.', 'It', 'is', 'for', 'us', 'the', 'living,', 'rather,', 'to', 'be', 'dedicated', 'here', 'to', 'the', 'unfinished', 'work', 'whic
h', 'they', 'who', 'fought', 'here', 'have', 'thus', 'far', 'so', 'nobly', 'advanced.', 'It', 'is', 'rather', 'for', 'us', 'to', 'be', 'here', 'dedicate
d', 'to', 'the', 'great', 'task', 'remaining', 'before', 'us', '--', 'that', 'from', 'these', 'honored', 'dead', 'we', 'take', 'increased', 'devotion',
'to', 'that', 'cause', 'for', 'which', 'they', 'gave', 'the', 'last', 'full', 'measure', 'of', 'devotion', '--', 'that', 'we', 'here', 'highly', 'resolv
e', 'that', 'these', 'dead', 'shall', 'not', 'have', 'died', 'in', 'vain', '--', 'that', 'this', 'nation,', 'under', 'God,', 'shall', 'have', 'a', 'ne
w', 'birth', 'of', 'freedom', '--', 'and', 'that', 'government', 'of', 'the', 'people,', 'by', 'the', 'people,', 'for', 'the', 'people,', 'shall', 'no
t', 'perish', 'from', 'the', 'earth.']
```

c. What is the length of `speech_list`?

   d. However, `speech_list` now contains non-words such as '--'. Create another function called `speech_to_list_better()` that uses the `append()` method to create a list by examining words one by one and appending them to the output list only if they aren't '--'. Store this in a list called `speech_list_better` and report its length.

   e. Create a function called `unique_words()` that takes `speech_list_better` and outputs a list containing only the unique words in Lincoln's speech. Store this in a list called `speech_list_unique` and report its length. Note here that (i) Capitalization doesn't affect uniqueness, i.e., "we" and "We" are the same word ("we"), and (ii) Surrounding punctuation doesn't affect uniqueness, i.e., "here," "here." "Here" and "here" are all the same word ("here").

4. **Object Oriented Programming.** The Euclidean distance between two d-dimensional vectors $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]$ and $\boldsymbol{y} = [y_1, y_2, \ldots, y_d]$ is given by $D(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$. Additionally, the (vector) sum of $\boldsymbol{x}$ and $\boldsymbol{y}$ is given by $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y} = [(x_1 + y_1), (x_2 + y_2), \ldots, (x_d + y_d)]$. Finally, the length of a vector (also known as its 2-norm) is given by $\|\boldsymbol{x}\| = \sqrt{\sum_{i=1}^{d} x_i^2}$.

a. Define a class called `Vector` with two attributes, dimension and coordinates. The former, dimension, stores the dimension of the space in which this vector object is defined, and the latter stores the vector object's coordinates. Both of these need to be specified during initialization and have default values of 2 and [0.0, 0.0] respectively.
b. Next, define two methods `distance()` and `vector_sum()`. The `distance()` method takes another Vector object, say, y and returns the Euclidean distance between the current `Vector` and y.

   The `vector_sum()` method also takes a Vector object, say y again, and returns another `Vector` object whose coordinates are given by the vector sum of the two vectors.
c. Finally, what does the "`__str__()`" method do? Define a "`__str__()`" method that prints the coordinates of the `Vector` object.
d. Using only the above answer the following questions. Suppose $\boldsymbol{x} = [1.0, -1.0, 3.0, 5.0, -2.2]$ and $\boldsymbol{y}$ is the origin, how far is $\boldsymbol{x}$ from $\boldsymbol{y}$? If $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$, what is the length of the vector $\boldsymbol{z}$? Without explicitly accessing any attributes of $\boldsymbol{z}$, print its coordinates.