Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [23]:
```
NAME = ""
COLLABORATORS = ""
```

CSE 30 Spring 2022 - Homework 1

Instructions

Please disregard the YOUR NAME and COLLABORATORS above. They are put there atomatically by the grading tool. Here's a quick explanation on how to turn in your work:

Submitting your work

To submit your work:

- First, click on "Runtime > Restart and run all", and check that you get no errors. This enables you to catch any error you might have introduced, and not noticed, due to your running cells out of order.
- Second, download the notebook in .ipynb format (File > Download .ipynb) and upload the .ipynb file to this form.
- This homework is due at **11:59pm on Wednesday, 6 April 2022**.

You can submit multiple times; the last submission before the deadline is the one that counts.

Homework format

For each question in this notebook, there is:

- A text description of the problem.
- One or more places where you have to insert your solution. You need to complete every place marked:

  `# YOUR CODE HERE`

  and you should not modify any other place.

- One or more test cells. Each cell is worth some number of points, marked at the top. You should not modify these tests cells. The tests pass if no error is printed out: when there is a statement that says, for instance:

  ```
  assert x == 2
  ```

  then the test passes if `x` has value 2, and fails otherwise. You can insert a `print(x)` (for this case!) somewhere if you want to debug your work; it is up to you.

Notes:

- Your code will be tested both according to the tests you can see (the `assert` statements you can see), *and* additional tests. This prevents you from hard-coding the answer to the particular questions posed. Your code should solve the *general* intended case, not hard-code the particular answer for the values used in the tests.
- **Please do not delete or add cells!** The test is autograded, and if you modify the test by adding or deleting cells, even if you re-add cells you delete, you may not receive credit.
- **Please do not import modules that are not part of the standard library.** You do not need any, and they will likely not available in the grading environment, leading your code to fail.
- **If you are inactive too long, your notebook might get disconnected from the back-end.** Your work is never lost, but you have to re-run all the cells before you continue.
- You can write out print statements in your code, to help you test/debug it. But remember: the code is graded on the basis of what it outputs or returns, not on the basis of what it prints.
- **TAs and tutors have access to this notebook,** so if you let them know you need their help, they can look at your work and give you advice.

Grading

Each cell where there are tests is worth a certain number of points. You get the points allocated to a cell only if you pass *all* the tests in the cell.

The tests in a cell include both the tests you can see, and other, similar, tests that are used for grading only. Therefore, you cannot hard-code the solutions: you really have to solve the essence of the problem, to receive the points in a cell.

Code of Conduct

- Work on the test yourself, alone.

- You can search documentation on the web, on sites such as the Python documentation sites, Stackoverflow, and similar, and you can use the results.
- You cannot share your work with others or solicit their help.

Question 1: Encoding and Decoding Text

We will write simple functions to encode and decode text.

The decoding function

The decoding function takes as input a list of n words $[w_0, w_1, \ldots, w_{n-1}]$, and a list of m integers $[k_0, k_1, \ldots, k_{m-1}]$, where each integer is in the range $[0, \ldots, n-1]$.

The function must output the string $w_{k_0}\ w_{k_1}\ w_{k_2}\ \ldots w_{k_m}$, where each number has been replaced with the word in the corresponding position in the list, and words are joined by spaces.

For example, assume the function is called with:

```
textdecode(["a", "bird", "seed", "eats"], [0, 1, 3, 0, 2])
```

Then the output should be

```
"a bird eats a seed"
```

because word 0 in the list is "a", word 1 is "bird", word 3 is "eats", word 0 is "a", and word 2 is "seed".

Here, we remind you that if you have a list of word, you can join them using spaces to form a string like this:

```
" ".join(["coffee", "is", "good"])
```

which produces

```
"coffee is good"
```

You have to write the decode function.

In [24]:
```
def textdecode(words, indices):
    """Decodes, producing a sentence in which each index is replaced
    by the word in the corresponding position in words, and the words
    are joined via spaces."""
    wordOrder = []
    for i in indices:
      wordOrder.append(words[i])
```

```
    phrase = " ".join(wordOrder)
    return phrase
```

In [25]:
```
# You can use this cell to test your code.

print(textdecode(["ha"], [0, 0, 0]))
```

```
ha ha ha
```

In [26]:
```
# 10 points: decoding function.

assert textdecode(["a", "bird", "seed", "eats"], [0, 1, 3, 0, 2]) == "a bird eats a
assert textdecode(["ha"], [0, 0, 0]) == "ha ha ha"
assert textdecode(["be", "to", "not", "or"], [1, 0, 3, 2, 1, 0]) == "to be or not t
```

Encoding function

The encoding function works in the opposite way. Given a sentence `s`, it first splits `s` according to spaces, via `s.split()`, obtaining a list of words. It then removes duplicates from this list of words, and returns:

- The list of words, with duplicates removed;
- A list obtained by replacing each word with the index of the word in the above list.

For instance, if the input is:

```
"i need a coffee"
```

then the words are `"i"`, `"need"`, `"a"`, `"coffee"`, and there are no duplicates. The output will then be:

```
["i", "need", "a", "coffee"], [0, 1, 2, 3]
```

If the input is

```
"yo ho ho and a bottle of rum"
```

the output can be

```
["yo", "ho", "and", "a", "bottle", "of", "rum"], [0, 1, 1, 2, 3, 4,
5, 6]
```

or if you prefer,

```
["a", "bottle", "of", "rum", "and", "yo", "ho"], [5, 6, 6, 4, 1, 2,
3]
```

which would decode to the same string.
You have to write the encode function.

In [27]:
```
def textencode(s):
    """Encode the string s, dividing it into words, and returning a list
    of unique words, and the list of indices of each word of s in the word list,
    in order."""
    words = s.split(" ")
    encoded = []
    order = []
    for word in words:
      if word not in encoded:
        encoded.append(word)
    for word in words:
      order.append(encoded.index(word))
    return encoded, order
```

In [28]:
```
# You can use this cell to test your code.

textencode("yo ho ho and a bottle of rum")
```

Out[28]:
```
(['yo', 'ho', 'and', 'a', 'bottle', 'of', 'rum'], [0, 1, 1, 2, 3, 4, 5, 6])
```

In [29]:
```
# some simple tests

assert textencode("a") == (["a"], [0])
assert textencode("ho ho ho") == (["ho"], [0, 0, 0])
```

In [30]:
```
# 10 points: encoding / decoding tests

import random

words = ["a", "bi", "ci", "di", "e", "effe", "gi"]
for _ in range(1000):
    k = random.randint(0, 10)
    s = " ".join(random.choices(words, k=k))
    ws, ix = textencode(s)
    assert len(set(ws)) == len(set(ix))
    t = textdecode(ws, ix)
    assert s == t
```

Question 2: Numerical Dictionaries

This question considers *numerical dictionaries*, which map keys to integers (rather than to arbitrary values). These dictionaries can be used to count occurrences; for instance, the dictionary {'a': 1, 'b': 3} indicates that 'a' occurred once, and 'b' three times.

Write a function `add_dicts` that, given two numerical dictionaries, computes the numerical dictionary resulting from their sum. For instance, `add_dict({'a': 1, 'b': 3}, {'b': 2, 'c': 3})` should give `{'a': 1, 'b': 5, 'c': 3}` as result.

In [31]:
```
def add_dicts(d1, d2):
    new_dict = {}
    for key in d1:
      if key in new_dict:
        new_dict[key] = new_dict[key] + d1[key]
      if key not in new_dict:
        new_dict.setdefault(key, d1.get(key))

    for key in d2:
      if key in new_dict:
        new_dict[key] = new_dict[key] + d2[key]
      if key not in new_dict:
        new_dict.setdefault(key, d2.get(key))
    return new_dict
```

In [32]:
```
# You can use this cell to test your code.

d1 = {'a': 1, 'b': 3}
d2 = {'a': 3, 'b': 1}
add_dicts(d1, d2)
```

Out[32]:
```
{'a': 4, 'b': 4}
```

In [33]:
```
# 10 points: When the keys are the same.

assert add_dicts({'a': 1, 'b': 2}, {'a': 3, 'b': 1}) == {'a': 4, 'b': 3}
# And empty.
assert add_dicts({}, {}) == {}
```

In [34]:
```
# 10 points: When the keys are different.

assert add_dicts({'a': 1, 'c': 2}, {'a': 3, 'b': 1}), {'a': 4, 'b': 1, 'c': 2}
assert add_dicts({'a': 3, 'b': 1}, {'a': 1, 'c': 2}), {'a': 4, 'b': 1, 'c': 2}
```

Question 3

Implement a function `ordered_substring` that takes as input two strings, s and t. The function returns True if and only if the characters of s can be found, in the order in which they are in s, as characters of t. In other words, the function returns True if and only if it is possible to remove some characters from t, so that the remaining ones form the string s.

For example:

- `ordered_substring("abc", "a123b4b523bc")` should return True, because `"a123b4b523bc"` contains the characters `"abc"` in that order, taking its 1st, 5th, and last character.
- Instead, `ordered_substring("abc", "a123c4b523b")` should return False, because `"a123c4b523b"` does not contain a "c" after a "b".
- `ordered_substring("abb", "a123c4b523f")` should also return False, because there is no `"b"` after the first `"b"` in `"a123c4b523f"`.

In [35]:
```
def ordered_substring(s, t):
    """Returns True if s is an ordered substring of t, and False otherwise."""
    ordered_list = []
    string = ""
    test = []
    for letter in s:
      test.append(letter)
    for letter in t:
      if letter in s:
        ordered_list.append(letter)
    for i in test:
      if i in ordered_list:
        string = string + i
        del ordered_list[0:ordered_list.index(i)+1]

    if s == string:
      return True
    else:
      return False
```

In [36]:
```
# You can use this cell to test your code.

ordered_substring('abbc', 'abc')
```

Out[36]: False

In [37]:
```
# 10 points: Let us first test that the code works for empty and one-character stri

assert ordered_substring('', 'abc')
assert ordered_substring('a', 'cabd')
assert ordered_substring('d', 'cabd')
assert not ordered_substring('f', 'cabd')
assert not ordered_substring('a', '')
assert ordered_substring('', '')
```

In [38]:
```
# 10 points: Some other simple cases.

assert ordered_substring('ab', 'abcd')
assert ordered_substring('ac', 'abcd')
assert ordered_substring('bd', 'abcd')
assert not ordered_substring('ba', 'abcd')
assert not ordered_substring('af', 'abcd')
assert ordered_substring('123', '12345')
```

In [39]:
```
# 10 points: General cases.

assert ordered_substring('abc', 'abbbeec')
assert not ordered_substring('abbc', 'abc')
assert ordered_substring('aa', 'bacad')
assert ordered_substring('aaba', 'ababa')
```

Question 4: Elections

Write an `Election` class, used to record votes. The class has the following methods:

- `vote(c)` : records a vote for candidate `c` .
- `total_votes()` : returns the total number of votes.
- `winner()` : returns the set of candidates with maximal votes. Normally there is only one, but there could be a tie, in which case you return the set of tied candidates. Return `None` if no votes have been cast.
- `margin()` : returns the margin of victory, that is, the number of votes separating the fist from the second candidate. This can be 0, in case of a tie. If there is only one candidate, you can answer with the number of votes for the candidate. If no votes have been cast, return `None` .

We provide the initializer method; the idea is to store the election status as a dictionary mapping candidate name to number of votes received.

In [40]:
```
class Election(object):
    """Implements the election class."""

    def __init__(self):
        # Dictionary mapping voters to the number of votes received.
        self.votes = {}
        self.numWins = 0

    def vote(self, c):
        """Records a vote for candidate c."""
        if c in self.votes:
          self.votes[c] += 1
        else:
          self.votes.update({c: 1})

    def total_votes(self):
        """Returns the total number of votes cast."""
        votes = self.votes.values()
        num_votes = 0
        for vote in votes:
          num_votes += int(vote)
        return num_votes

    def winner(self):
        """Returns the list of winners of the election."""
        temp = self.votes.copy()
        winner = set()
        duplicate = True
        if temp == {}:
          return None
        while duplicate == True:
          highest = max(temp, key=temp.get)
          win = highest
          winner.add(win)
          temp.pop(highest)
          highest = max(temp, key=temp.get)
          if self.votes[win] > self.votes[highest]:
```

```
            duplicate = False
        self.numWins = len(winner)
        return winner

    def margin(self):
        """Returns the margin of victory of the winner."""
        if self.votes == {}:
          return None
        if self.numWins > 1:
          return 0
        temp = self.votes.copy()
        highest_key = max(temp, key=temp.get)
        win = self.votes[highest_key]
        temp.pop(highest_key)
        second_key = max(temp, key=temp.get)
        second = self.votes[second_key]
        return win - second
```

In [41]:
```
# You can use this cell to test your code.

e = Election()

e.vote('a')
e.vote('b')
e.vote('a')

print(e.winner())

e.vote('c')
e.vote('b')

print(e.winner())
```

```
{'a'}
{'a', 'b'}
```

In [42]:
```
# 10 points: Tests for total_votes method.

e = Election()
assert e.total_votes() == 0

e.vote('a')
e.vote('b')
e.vote('a')
assert e.total_votes() == 3

e.vote('c')
e.vote('b')
assert e.total_votes(), 5
```

In [43]:
```
# 10 points: Tests for winner method.

e = Election()
```

```
assert e.winner() == None

e.vote('a')
e.vote('b')
e.vote('a')
assert set(e.winner()) == {'a'}

e.vote('c')
e.vote('b')
assert set(e.winner()) == {'a', 'b'}
```

In [44]:
```
# 10 points: Tests for margin method.

e = Election()
assert e.margin() == None

e.vote('a')
e.vote('b')
e.vote('a')
assert e.margin() == 1

e.vote('c')
e.vote('b')
assert e.margin() == 0
```

In [ ]:
```

```