Homework 5: Fall 2020

Due Oct 5, 4PM

Fill in your name

In [301]:
```
first_name = "Shariq"
last_name = "Jamil"

assert(len(first_name) != 0)
assert(len(last_name) != 0)
```

Problem 1: Inorder

Take a list of elements, and decide if the elements are in ascending order.

The list may contain integers or strings, but will contain only one type of value.

```
def inorder(lst: List) -> bool:
```

Examples:

The list

```
[1, 4, 9, 13]
```

is in order. However

```
['one', 'two', 'three', 'four']
```

is not in order, as 'three' comes before 'two' in the dictionary

Fill in your function definition in the cell below.

In [317]:
```
'''
A function that takes in a list of
integers or strings. If the list is not
in order it returns False, otherwise True
'''
def inorder(lst):
    # counter to keep track of where we are in the list
    count = 0

    # if there are enough objs to compare
    if(len(lst) > 1):
        # traverse through list until the end
        while count != (len(lst)-1):
            # compare with the next object
            if(lst[count] > lst[count+1]):
                # the current object is larger (numerically/alphabetic
ally) than the next, abort
                return False
            # continue to the next obj, increment counter
            count+= 1
    # the list is in order
    return True
```

Test case for inorder()

In [303]:
```
def validate_inorder():
    assert inorder([1, 4, 9, 13]), "List is inorder"
    assert inorder([1]), "List is inorder"
    assert inorder([]), "List is inorder"
    assert inorder(['one', 'ten', 'three', 'two']), "List is inorder"

    assert not inorder([3, 1, 4]), "3 appears before 1"
    assert not inorder([3, 2, 1]), "3 appears before 2"
    assert not inorder([1, 4, 9, 13, 12]), "13 appears before 12"
    assert not inorder(['one', 'two', 'three', 'four']), "two appears
before three"

    print('Sucess!')

validate_inorder()
```

```
Sucess!
```

Problem 2: Sum of Two

Write a function that takes an integer target k and a list of integers, and decides if you can represent k as the sum of two different numbers in the list.

```
def sum_of_two(k: int, lst : List[int]) -> bool:
```

Examples:

```
sum_of_two(17, [1, 15, 3, 4, 5, 6, 7, 2])
```

returns True, as 17 = 15 + 2

```
sum_of_two(4, [1, 2])
```

returns False, as you cannot reuse the 2, and 4 is not 2 + 1.

Fill in your function definition in the cell below.

In [304]:
```
'''
A function that takes in an integer and a list of
integers. If two different integers in the list
add up to the given integer, the function returns True.
If not, it returns False
'''
def sum_of_two(k, lst):
    # keep track of which obj is being evaluated in both loops
    index1 = 0;
    index2 = 0;

    # compare each integer
    for num1 in lst:
        # to every integer in the list
        for num2 in lst:
            # if we are not comparing the object to itself and the sum
adds up to 'k'
            if ((index1 != index2) and (num1+num2 == k)):
                # pass
                return True
            # no match found, increment index for inner loop
            index2+=1
        # no match found, reset inner loop index
        index2 = 0
        # increment outer loop index
        index1+= 1
    return False
```

Test cases for sum of two

In [305]:
```
assert not sum_of_two(0, []), "Empty List"
assert not sum_of_two(3, [3]), "Singleton list"
assert sum_of_two(3, [1, 2]), "3 = 1 + 2"
assert sum_of_two(17, [10, 15, 3, 7]),  "17 = 10 + 7"
assert sum_of_two(4, [2, 2]), "4 = 2 + 2"
assert sum_of_two(4, [0, 4]), "4 = 0 + 4"
assert sum_of_two(17, [1, 15, 3, 4, 5, 6, 7, 2]), "17 = 15 + 2"

assert not sum_of_two(17, [10, 15, 4, 8]), "Cannot write 17 as sum of
elements"
assert not sum_of_two(4, [1, 2]), "Can't use the same 2 twice"

print('Sucess')
```

```
Sucess
```

Problem 3: Hamming Distance

The Hamming distance between two strings is the number of places where the strings don't agree.

We consider 'A' and 'a' to be the same letter.

```
def hamming_distance(word1: str, word2: str) -> int:
```

Examples:

```
hamming_distance('sugar', 'spice') = 4
```

as the two strings differ in every spot but the first.

```
hamming_distance("GGACG", "GGTCG") == 1
```

as the two strings only differ in the third place: A != T.

```
hamming_distance("tag", "GAT") == 2
```

as the strings differ in the first and third place. We treat 'a' and 'A' as equal.

```
hamming_distance("hot", "cold")
```

is not defined, as the strings have different lengths.

If the strings have different lengths, your function should throw an ValueError exception with text describing the problem in your own words

Fill in your function definition in the cell below.

In [306]:
```
# Return the number of differences
# Takes two strings, return non-negative integer
# Throws ValueError if the strings have different length
#
def hamming_distance(strand_a, strand_b):
    # throw exception if the two strings have different lengths
    if(len(strand_a) != len(strand_b)):
        raise ValueError('The two given strands must be of the same le
ngth')

    counter = 0
    distance = 0

    # traverse through first string
    while counter <= len(strand_a)-1:
        # compare the lowercase version of string at the same index
        if(strand_a[counter].lower() != strand_b[counter].lower()):
            # match not found, increment distance
            distance += 1
        # move on to next index
        counter += 1
    # return total distance
    return distance
```

In [307]:
```
### Test case for hamming_distance()

def test_hamming():
    assert hamming_distance("A", "A") == 0, "Same string"
    assert hamming_distance("GGACTGA", "GGACTGA") == 0, "Same string"
    assert hamming_distance("A", "G") == 1, "Differ in every place"
    assert hamming_distance("AG", "CT") == 2, "Differ in every place"
    assert hamming_distance("AT", "CT") == 1, "Differ in first place"
    assert hamming_distance("GGACG", "GGTCG") == 1, "Differ in third p
lace"
    assert hamming_distance("ggACG", "GGtCG") == 1, "Differ in third p
lace"
    assert hamming_distance("GGACG", "ggtCG") == 1, "Differ in third p
lace"
    assert hamming_distance("ACCAGGG", "ACTATGG") == 2, "Differ in two
places"
    assert hamming_distance("AAG", "AAA") == 1, "Differ in third plac
e"
    assert hamming_distance("AAA", "AAG") == 1, "Differ in third plac
e"
    assert hamming_distance("TAG", "GAT") == 2, "Differ in first and t
hird place"
    assert hamming_distance("GATACA", "GCATAA") == 4, "Differ in four
places"
    assert hamming_distance("GGACGGATTCTG", "AGGACGGATTCT") == 9, "Dif
fer in nine places"

    return 'Success'

test_hamming()
```

Out[307]:
```
'Success'
```

In [308]:
```
# Your function should throw an ValueError exception if the strings ha
ve different lengths
#
# If it doesn't, I will raise an exception
#
try:
    hamming_distance("AATG", "AAA")
    assert 1 == 2, "You were supposed to raise an Exception!"
except ValueError:
    print("Success")
except:
    assert 1 == 2, "You were supposed to raise an ValueError Exceptio
n!"
```

```
Success
```

Problem 4: Find Reversals

Write a function that takes a list, and returns a list representing each word whose reverse is also in the list.

```
def find_reversals(lst: List[str]) -> List[str]:
```

Each pair, such as 'abut', 'tuba', should be represented by the first element encountered. Don't report the same pairs twice.

Don't list palindromes.

Fill in your function definition in the cell below.

In [309]:
```
'''
This function takes in a list and returns a list of
each word whose reverse is also in the list

Same pairs and palindromes are not reported
'''
def find_reversals(lst):
    # init return list
    reverse_list = []
    # outer loop
    for word1 in lst:
        # reverse of word - what we are looking for
        reverse = word1[::-1]
        # inner loop to compare each string to
        for word2 in lst:
            # compare lowercase versions and make sure its not a palin
drome
            if(reverse.lower() == word2.lower() and reverse.lower() !=
word1.lower()):
                # if the word or its pair are not already in list
                if(word1.lower() not in reverse_list and reverse.lower
() not in reverse_list):
                    # add to return list
                    reverse_list.append(word1.lower())
                    # move on to next word if match was found
                    break
    return reverse_list
```

Test cases for find_reversals()

In [310]:
```
assert find_reversals(['art', 'Rat', 'Radar', 'scam', 'tar', 'vista'])
== ['rat']
assert find_reversals(['abut', 'Rat', 'Radar', 'tuba']) == ['abut']
assert find_reversals(['art', 'Rat', 'Radars', 'scam', 'tartars', 'vis
ta']) == []

assert find_reversals(['art', 'tuba', 'Rat', 'Radar', 'rat', 'radar',
'abut', 'tar', 'tar']) == ['tuba', 'rat']
assert find_reversals(['art', 'tuba', 'Rat', 'Radar', 'tar', 'tar', 'r
at', 'radar', 'abut']) == ['tuba', 'rat']

assert find_reversals(['Radar']) == []
assert find_reversals(['test']) == []
assert find_reversals([]) == []

print('Success!')
```

```
Success!
```

Problem 5: Find reversals in the dictionary

Write a program that finds the reversals in Downey's word list.

List each pair only once, and only report the first word: List 'abut', but not 'tuba'

Do not list palindromes.

```
def find_reversals_in_file(fileName: str) -> List[str]:
```

If you try to open a file that does not exist, you should catch a FileNotFoundError and print an error message in your own words

Fill in your function definition in the cell below.

In [311]:
```
'''
This function takes in a list and returns a list of
each word whose reverse is also in the list

It is assumed that a sorted list of distinct strings is being submitte
d.
'''
def find_reversals_dict(lst):
    # init return list
    reverse_list = []
    # convert list to set for faster searching
    dict_set = set(lst)
    # traverse through each string
    for word1 in lst:
        # trimmed reverse of word - what we are looking for
        word1 = word1.lower()
        reverse = word1[::-1]
        # look for reversed string in set
        if(reverse in dict_set
           # do not add pairs that are already in the return list
            and reverse not in reverse_list and word1 not in reverse_l
ist
           # do not add palindromes
            and reverse != word1):
                    # if found, add to return list
                    reverse_list.append(word1.lower())
    return reverse_list

def find_reversals_in_file(filename):
    try:
        # open given file
        with open(filename, 'r') as words:
            # trim given lines
            text = words.read().splitlines()
            # pass list of strings into reversal finder
            rev_list = find_reversals_dict(text)
            return rev_list
    except FileNotFoundError:
        print('File not found. Please verify file location and name')
```

Call your function in the cell below. You may change the path to point to your copy of words.txt

In [312]:
```
# Call your function here
lst = find_reversals_in_file("words.txt")

print(f"There were {len(lst)} reversals")

for word in lst[:10]:
    print(word)
```

```
There were 397 reversals
abut
ad
ados
agar
agas
agenes
ah
aider
airts
ajar
```

Call your function on a file that doesn't exist You should catch the exception and print a message in your own words

In [313]:
```
# Call your function here on a file that doesn't exist
#
lst = find_reversals_in_file("mxyzptlk.txt")
```

```
File not found. Please verify file location and name
```

Problem 6: Find Python files

Starting with Downey's walk.py, write a function find_python_files() to return a list of all Python files below a directory in the file system.

```
def find_python_files(dirName: str) -> List[str]:
```

When I call it on my directory 'Python/Programs', I get a list like this:

```
./day4/cross.py
./day4/hanoi.py
./day4/isvowel.py
./day4/Koch.py
./day4/dragon.py
./day3/binary_search.py
./day3/file2.py
./day3/reverse.py
./day3/longwords2.py
./day3/paint.py
./day3/file3.py
...
```

Incude in your notebook output an example with at least this level of complexity: multiple levels and multiple directories.
(You may need to create some directories and copy some file around to achieve that.)

define your function below

In [314]:
```
'''
This function takes in a directory name and
returns a list of all python files below a point in the file system
'''
import os
import sys

# global var to track python files
python_files = []

def walk(dirname: str):
    # Walk over the files in this directory
    for name in os.listdir(dirname):
        # Construct a full path
        path = os.path.join(dirname, name)
        # if a file is found
        if os.path.isfile(path):
            # if extension is py
            if(path[-3:] == '.py'):
                # add to list
                python_files.append(path)
        # traverse directory
        else:
            walk(path)

def find_python_files(dirname: str):
    # call walk function to get list of python files below a dir
    walk(dirname)
    return python_files
```

Call your function below. You may change the directory to find your python files.

In [315]:
```
lst = find_python_files('..')

for w in lst:
    print(w)
```

```
..\Homework 3\binary_search.py
..\Homework 3\binary_search_test.py
..\Homework 3\deliverables\freestyle.py
..\Homework 3\deliverables\grid.py
..\Homework 3\deliverables\honeycomb.py
..\Homework 3\deliverables\pentagram.py
..\Homework 5\Programs\Prog1.py
..\Homework 5\Programs\Prog2.py
..\Homework 5\Programs\Prog3 - Copy (10).py
..\Homework 5\Programs\Prog3 - Copy (11).py
..\Homework 5\Programs\Prog3 - Copy (12).py
..\Homework 5\Programs\Prog3 - Copy (13).py
..\Homework 5\Programs\Prog3 - Copy (14).py
..\Homework 5\Programs\Prog3 - Copy (15).py
..\Homework 5\Programs\Prog3 - Copy (16).py
..\Homework 5\Programs\Prog3 - Copy (17).py
..\Homework 5\Programs\Prog3 - Copy (18).py
..\Homework 5\Programs\Prog3 - Copy (19).py
..\Homework 5\Programs\Prog3 - Copy (2).py
..\Homework 5\Programs\Prog3 - Copy (3).py
..\Homework 5\Programs\Prog3 - Copy (4).py
..\Homework 5\Programs\Prog3 - Copy (5).py
..\Homework 5\Programs\Prog3 - Copy (6).py
..\Homework 5\Programs\Prog3 - Copy (7).py
..\Homework 5\Programs\Prog3 - Copy (8).py
..\Homework 5\Programs\Prog3 - Copy (9).py
..\Homework 5\Programs\Prog3 - Copy.py
..\Homework 5\Programs\Prog3.py
..\Homework 5\walk.py
```

Post Mortem

How long did it take you to solve this problem set?

Did anything confuse you or cause difficulty?

In [316]:
```
# Enter your thoughts
# 10 hours. Nothing confusing, between piazza, video preview and docum
entation I was able to gather everything I needed.
```