Homework 5 Solutions

Fill in your name

In [ ]:

```
first_name = ""
last_name = ""

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 [1]:

```
def inorder(lst):
    for i in range(1, len(lst)):
        if lst[i-1] > lst[i]:
            return False

    return True
```

Test case for inorder()

In [2]:

```
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

    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 [3]:

```
def sum_of_two(k, lst):
    for i in range(len(lst)):
        val = lst[i]
        if k - val in lst[i+1:]:
            return True
    return False
```

Test cases for sum of two

In [4]:

```
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
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 [23]:

```
# 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):
    if len(strand_a) != len(strand_b):
        raise ValueError('Strings should have the same length')

    dist = 0
    strand_a = strand_a.lower()
    strand_b = strand_b.lower()
    for i in range(len(strand_a)):
        if strand_a[i] != strand_b[i]:
            dist = dist + 1
    return dist
```

When we know a bit more, we will be able to rewrite this to be more Pythonic, as below

In [ ]:

```
# 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):
    if len(strand_a) != len(strand_b):
        raise ValueError('Strings should have the same length')

    return sum(a != b for a, b in zip(strand_a.lower(), strand_b.lower
```

In [37]:

```
### 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
    assert hamming_distance("ggACG", "GGtCG") == 1, "Differ in third p
    assert hamming_distance("GGACG", "ggtCG") == 1, "Differ in third p
    assert hamming_distance("ACCAGGG", "ACTATGG") == 2, "Differ in two
    assert hamming_distance("AAG", "AAA") == 1, "Differ in third place
    assert hamming_distance("AAA", "AAG") == 1, "Differ in third place
    assert hamming_distance("TAG", "GAT") == 2, "Differ in first and t
    assert hamming_distance("GATACA", "GCATAA") == 4, "Differ in four
    assert hamming_distance("GGACGGATTCTG", "AGGACGGATTCT") == 9, "Dif

    return 'Success'

test_hamming()
```

Out[37]:

```
'Success'
```

In [25]:

```
# Your function should throw an ValueError exception if the strings ha
#
# 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 Exceptior
```

```
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 [16]:
```
def find_reversals(lst):
    lst = [word.lower() for word in lst]
    res = []
    for word in lst:
        if word not in res:
            rev = word[::-1]
            if rev != word and rev not in res and rev in lst:
                res.append(word)
    return res
```

Test cases for find_reversals()

In [15]:
```
assert find_reversals(['art', 'Rat', 'Radar', 'scam', 'tar', 'vista'])
assert find_reversals(['abut', 'Rat', 'Radar', 'tuba']) == ['abut']
assert find_reversals(['art', 'Rat', 'Radars', 'scam', 'tartars', 'vis

assert find_reversals(['art', 'tuba', 'Rat', 'Radar', 'rat', 'radar',
assert find_reversals(['art', 'tuba', 'Rat', 'Radar', 'tar', 'tar', 'r

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 [17]:

```
def read_file(filename):
    res = []

    try:
        with open(filename, 'r') as words:
            for word in words:
                res.append(word.strip())

        return res

    except FileNotFoundError:
        print(f"Could not find file: {filename}")
    except:
        print(f"Could not open file: {filename}")

    return []



# Enter your function  here
def find_reversals_in_file(filename):
    return find_reversals(read_file(filename))
```

Call your function in the cell below.

In [18]:

```
lst = find_reversals_in_file("../../Programs/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 here on a file that doesn't exist

In [19]:

```
# Call your function here on a file that doesn't exist
# This will throw an exception:
#     you should catch the exception, and print a message in your own
#
lst = find_reversals_in_file("mxyzptlk.txt")
```

```
Could not find file: mxyzptlk.txt
```

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 [1]:

```
import os

## Start with walk
def walk(dirname: str):
    "Perform a recursive traverse of directories"

    res = []

    # Walk over the files in this directory
    for name in os.listdir(dirname):

        # Construct a full path
        path = os.path.join(dirname, name)

        # print filenames, and traverse directories
        if os.path.isfile(path):
            res.append(path)
        else:
            res = res + walk(path)

    return res
```

I changed three lines

In [6]:

```
import os

## Start with walk
def find_python_files(dirname: str):           # Change name
    "Perform a recursive traverse of directories"

    res = []

    # Walk over the files in this directory
    for name in os.listdir(dirname):

        # Construct a full path
        path = os.path.join(dirname, name)

        # print filenames, and traverse directories
        if os.path.isfile(path):
            if path.endswith('.py'):          # Change
                res.append(path)
        else:
            res = res + find_python_files(path)  # Change name

    return res
```

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

```
In [11]: lst = find_python_files('../../Programs')

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

```
../../Programs/advanced/serialize/serialize.py
../../Programs/advanced/serialize/debug.py
../../Programs/advanced/serialize/load.py
../../Programs/tools/BuildStudent.py
../../Programs/tools/ReadEdFile.py
../../Programs/assignment1/prog1.py
../../Programs/AutoGrade/runProgs.py
../../Programs/AutoGrade/runTests.py
../../Programs/day4/hanoi2.py
../../Programs/day4/test_leapYear.py
../../Programs/day4/cross.py
../../Programs/day4/hanoi.py
../../Programs/day4/isvowel.py
../../Programs/day4/Koch.py
../../Programs/day4/dragon.py
../../Programs/day4/reverse.py
../../Programs/day4/leapyear.py
../../Programs/day4/vowels.py
../../Programs/day4/traverse.py
../../Programs/day4/read2.py
```

Post Mortem

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

Did anything confuse you or cause difficulty?

```
In [ ]: # Enter your thoughts
```