Homework 4

Fill in your name

In [89]:
```
# Fill in your name

first_name = 'Shariq'
last_name = 'Jamil'

assert len(first_name) != 0, "First name is blank"
assert len(last_name) != 0, "Last name is blank"
```

This assignment uses an idiom called Filtering

Filtering - running through a sequence of items, and testing each item with a Boolean function.

We filter your luggage at the airport. We look at each item in your suitcase, and stop you if you have a gun, explosives, or a bottle of water.

We use filtering to decide if we should let you board

We filter our reciepts after a trip. We can charge the meals and the room to the company, but can't charge the tickets to see Buddy Guy.

We filtering to find the sublist of items to pass on

Problem 1

is_vowel()

Is this character a vowel?

This function takes a string and returns a Boolean.

```
def is_vowel(ch: str) -> bool:
```

Fill in your function definition in the cell below.

In [90]:
```
'''
A function takes in a character and returns
True if the character is a vowel, False if not.
'''
def is_vowel(ch):
    # set char to lowercase so we don't have to test seperately for uppercase
    ch = ch.lower()
    # compare char to each vowel
    if(ch == 'a' or ch == 'e' or ch == 'i' or ch == 'o' or ch == 'u'):
        # return True if a match was found
        return True
    # no match was found
    return False
```

Test cases for is_vowel()

Run the cell below to call your function.

In [91]:
```
import string

def validate_vowel():
    for ch in 'aeiouAEIOU':
        assert is_vowel(ch), f"'{ch}' should be a vowel"

    for ch in string.ascii_lowercase:
        if ch not in 'aeiou':
            assert not is_vowel(ch), f"'{ch}' is not a vowel"
            assert not is_vowel(ch.upper()), f"'{ch.upper()}' is not a vowel"

    print('Success!')

validate_vowel()
```

```
Success!
```

Our Unit Test uses filtering

In this unit test, we run through a list of vowels looking for a counterexample.

Problem 2: has_all_vowels()

Does the string contain all the vowels?

This function takes a string and returns a Boolean.

```
def has_all_vowels(word: str) -> bool:
```

Fill in your function definition in the cell below.

In [92]:
```
'''
A function takes in a word and returns
True if the word contains all five vowels, False if not.
'''
def has_all_vowels(word):
    # initialize all five trackers as False
    a_found = False
    e_found = False
    i_found = False
    o_found = False
    u_found = False

    # check each character
    for ch in word:
        # compare lowercase versions
        ch = ch.lower()
        if(ch == 'a'):
            # if a match is found, mark as such
            a_found = True
        elif(ch == 'e'):
            e_found = True
        elif(ch == 'i'):
            i_found = True
        elif(ch == 'o'):
            o_found = True
        elif(ch == 'u'):
            u_found = True
    # if all vowels had a match, return True
    if(a_found and e_found and i_found and o_found and u_found):
        return True
    # otherwise return False
    else:
        return False
```

Test cases for has_all_vowels()

Run the cell below to call your function.

In [93]:
```
def validate_all_vowels():
    assert has_all_vowels('vexatious'), "'vexatious' has all the vowels"
    assert has_all_vowels('VEXATIOUS'), "'VEXATIOUS' has all the vowels"

    # Now remove one vowel at a time
    # Each test case should return False
    assert not has_all_vowels('vxatious'), "Missing 'e'"
    assert not has_all_vowels('vextious'), "Missing 'a'"
    assert not has_all_vowels('vexatous'), "Missing 'i'"
    assert not has_all_vowels('vexatius'), "Missing 'o'"
    assert not has_all_vowels('vexatios'), "Missing 'u'"

    print('Success!')

validate_all_vowels()
```

```
Success!
```

Problem 3: is_all_vowels()

Is every character in word a vowel?

This function takes a string and returns a Boolean.

```
def is_all_vowels(word: str) -> bool:
```

Fill in your function definition in the cell below.

In [94]:
```
'''
A function takes in a word and returns
False if the word contains any nonvowel, False if not.
'''
def is_all_vowels(word):
    for char in word:
        char = char.lower()
        if(char != 'a' and char != 'e' and char != 'i' and char != 'o' and char != 'u'):
            return False
    return True
```

Test case for is_all_vowels()

Run the cell below to call your function.

In [95]:
```
def validate_is_all_vowels():
    assert is_all_vowels('aie'), "'aie' is all vowels"
    assert is_all_vowels('Aie'), "'Aie' is all vowels"
    assert is_all_vowels('AeIoU'), "'AeIoU' is all vowels"

    assert not is_all_vowels('kaie'), "has a k"
    assert not is_all_vowels('akie'), "has a k"
    assert not is_all_vowels('aike'), "has a k"
    assert not is_all_vowels('aiek'), "has a k"

    print('Success!')

validate_is_all_vowels()
```

```
Success!
```

Problem 4: is_palindrome()

Write a function that decides if a word is a palindrome, like Radar or madam.

Is the word the same, read forwards or backwards?

Your function is_palindrome() should take a string and return a Boolean

```
def is_palindrome(word: str) -> bool:
```

Fill in your function definition in the cell below.

In [96]:
```
'''
A function takes in a word and returns
True if the word is a palindrome, False if not.
'''
def is_palindrome(word):
    # initialize string to hold reverse version of given word
    reverse = ''
    # iterate through each character
    for ch in word:
        # prepend string to create reversed string
        reverse = ch + reverse
    # compare lowercase versions of both strings
    if(reverse.lower() == word.lower()):
        # palindrome found, both strings matched
        return True
    else:
        # not a palindrome
        return False
```

Test cases for is_palindrome

Run the cell below to call your function.

In [97]:
```
def validate_palindrome():
    lst = ['A', 'radar', 'Radar', 'Ada', 'madaM']
    for word in lst:
        assert is_palindrome(word), f"'{word}' is a palindrome"

    lst = ['adar', 'loop', 'leal']
    for word in lst:
        assert not is_palindrome(word), f"'{word}' is not a palindrome"


    print('Success!')

validate_palindrome()
```

```
Success!
```

Problem 5: Crossword Puzzle

Write a function that detects five letter words that start with 'a' and end in 't'.

Your function is_match() should take a string and return a Boolean

```
def is_match(word: str) -> bool:
```

Fill in your function definition in the cell below.

In [98]:
```
'''
A function takes in a word and returns
True if the word has five letters and starts with 'a'
and ends in 't', False if not.
'''
def is_match(word):
    # word length check, starts with 'a' check and ends with 't' check
    if(len(word) == 5 and word[0] == 'a' and word[len(word)-1] == 't'):
        # passes checks
        return True
    else:
        # did not pass
        return False
```

Run the cell below to call your function.

In [99]:
```
def validate_match():
    for word in ['A', 'Radar', 'at', 'splat', 'adotp']:
        assert not is_match(word), f"'{word}' is not a match"

    for word in ['adopt', 'agent', 'argot', 'avast', 'adult']:
        assert is_match(word), f"'{word}' should be a match"

    print('Success!')

validate_match()
```

```
Success!
```

Problem 6: Find all matches in the Dictionary

Write a function that takes a Boolean function and a filename and prints the matches

Open the file words.txt, and print all five letter words that start with 'a' and end in 't'.

Your function find_match() should take a function and a filename and return a list of strings.

The function you pass in to find_match should take a string and return a Boolean.

We can use Python's Type Hints to describe the interface. This looks complicated, but this is a complicated idea.

```
f: Callable[[str], bool]
```

says: the parameter f is a function (Callable) that takes a string and returns a Boolean.

The return value is described by this:

```
-> List[str]:
```

which means "The function returns a list of strings".

The full description is:

```
from typing import List, Callable

    def find_match(f: Callable[[str], bool], filename: str) -> List[str]:
```

Fill in your function definition in the cell below.

In [100]:
```
# Write a function that uses your is_match() to filter the contents of words.txt

# Your function find_match takes a function: pass in your function is_match()
# find_match should return a list holding the words in the file words.txt that sat
ify the function passed in
def find_match(func, filename):
    # initialize list of words to return
    ret_list = []
    # parse file
    with open(filename, 'r') as words:
        # iterate through each line
        for line in words:
            # remove any starting/trailing spaces from the line
            line = line.strip()
            # run test
            if(func(line)):
                # test passed, add to list
                ret_list.append(line)
    return ret_list
```

Run the cell below to call your function. Be sure you have Downey's **words.txt** in the same directory as your notebook

In [101]:
```
# Call your function to find the matches

lst = find_match(is_match, "words.txt")

print(lst)
```

```
['abaft', 'abbot', 'abort', 'about', 'adapt', 'adept', 'admit', 'adopt', 'adult
', 'adust', 'afoot', 'afrit', 'agent', 'agist', 'aglet', 'alant', 'alert', 'alis
t', 'allot', 'aloft', 'ambit', 'ament', 'amort', 'anent', 'angst', 'apart', 'apo
rt', 'argot', 'arhat', 'armet', 'ascot', 'asset', 'atilt', 'audit', 'aught', 'av
ast', 'avert', 'await']
```

Problem 7: Find the palindromes in words.txt.

Use your function from Problem 6, but use is_palindrome(), the Boolean function you defined in problem 4, as the filter.

You should be able to use your work from problems 4 and 6 without changing them

Be sure you have Downey's **words.txt** in the same directory as your notebook. Print the number of palindromes in the file. Print the first 10 palindromes in the file.

In [102]:
```
# Write a Python fragment that uses your work from problems 2 and 4
'''
A function that performs a palindrome test on every word in
words.text and prints the number of palindromes in the text file
along with the first ten palindromes
'''
def find_palindromes():
    # runs palindrome test on every word and returns words that pass
    lst = find_match(is_palindrome, "words.txt")
    # print the number of palindromes
    print(len(lst))
    # counter for tracking how many words have printed
    counter = 1
    for word in lst:
        # for the first ten words
        if(counter <= 10):
            # print the word
            print(word)
        # append counter
        counter+=1
find_palindromes()
```

```
91
aa
aba
aga
aha
ala
alula
ama
ana
anna
ava
```

Post Mortem

How long did it take you to solve this problem set? Did anything confuse you or cause difficulty?

In [103]:
```
# Enter your thoughts
# It took me about two hours. No difficulties or issues
```