Name:________________________ **Partner:** ________________________

Python Activity 18: Designing Algorithms for Sequences

Awesome things we can do with our awesome list, string, and sequence operators!

Learning Objectives
Students will be able to:
Content:
- Define a **docstring** and **doctest**
- **Decompose Problems:** Identify the sub-problems within a given problem.
- **Encapsulate** smaller, repeated sub-solutions into **helper functions**
- Design **algorithms** to solve a given problem.

Process:
- Incorporate **docstrings** and **doctests** into our code
- Write code to iterate over nested sequences to collect specified information
- Use appropriately designed accumulator variables for given problems

Prior Knowledge
- Python concepts: lists, strings, for loops, nested lists, nested loops

Critical Thinking Questions:

1. Examine the sample code defining a list below.

Sample Code

```
dog_list =
["pixel howley","chelsea doret","artie q. jannen","sally albrecht","velma"]
```

a. Given a list, `dog_list`, we want to find all names that contain a certain letter, `character`, in `dog_list` using an **algorithm** that is generalizable to other lists of names. What might we have to keep track of in order to do this?

FYI: An ***algorithm*** is a sequences of generalizable steps to solve a particular problem.

b. Write out pseudocode for a generalizable algorithm that will identify which names in `dog_list` contains letter, `character`:

c. How might you adapt your approach to find only *first names* that contain `character`?

2. *Continuing on with our algorithm design…*

   a. Given a string, `name` we want to generate a substring representing only the *first name*. When given a name, similar to the elements in `dog_list` how do we know what is the first name?

   b. We want this code to work for all of the names in `dog_list`. What special cases might we have to consider?

   c. Complete the following function body to return the first name of a string `name`:

```
def get_firstname(name):
    """ returns the firstname in string, name
     >>> get_firstname("pixel howley")
     'pixel'
     """
    # initialize accumulator variable

    # look at each character in the name

        # if this character is a space, we're done!


        # otherwise, accumulate the character

    # return the name
```

> **FYI:** ***Docstrings*** are multiline comments that appear just under a *function header* but above the *function body* that describe what that function does. They are denoted with tripe-quotes (either single or double), and often include ***doctests*** which are example snippets of code to test the function in interactive python.

   d. Place a star next to the ***docstring*** in this example. Place a triangle next to the ***doctest***. What might be an additional good ***doctest*** for this function?

```
>>> get_firstname(__________________________________________)
# what should be returned?
```

3. Examine the code below, that finds all names that contain a certain letter, `character`, in `dog_list`:

Sample Code

```
def first_contains_character(name_list, char):
    """ Returns a list of names in name_list containing character"""
    result = []
    for name in name_list:
        if char in get_firstname(name):
            result = result + [name]
    return result
```

a. Trace through this function with the example function call `first_contains_character(["pixel howley","chelsea doret"], 't')`:

```
name_list = [_______________,_______________]      char = _____
______________________________________________________________________
result                   name             char in get_firstname(name)
```

b. What will be returned by the function call `first_contains_character(["pixel howley","chelsea doret"], 't')`?

c. We want to change this function to only return names that *start* with the given character, `char`. Circle what code would have to change. What code would you replace it with?

d. What would be a good ***doctest*** for this new function, `starts_character(..)`?

```
>>> starts_character(__________________________________________)
# what should be returned?
```

3. Now we'd like to gather two lists, one of the longest names in a `name_list`, and one of the shortest names. Here's an example ***doctest***:

```
>>> dl = ["pixel howley","chelsy doret","artie jannen","velma"]
>>> shortlong_names(dl)
[['velma'], ['chelsy doret', 'artie jannen']]
```

a. What *type* of object does `shortlong_names` return? _____________________

b. How might we access the *shortest* name in this returned result?

c. Why might `'velma'` be returned as a list of strings, rather than just a string?

d. Write pseudocode to explain your algorithm for the `shortlong_names(name_list)` function:

Convert your pseudocode to Python in a file after class, and see if it works! Fix any logic errors!

4. Now we'd like to write a function, `last_names(name_list)` that will return a list of all the lastnames in `name_list`. To do so, might consider writing a ***helper function***, `get_lastname(name)` which returns the last name from a `name` string, just as we did with `get_firstname`. However, there is a more generalizable solution that will work for retrieving first names, last names, *and* middle names. Observe the following example `name_list`:

```
name_list =
["pixel howley","chelsea doret","artie q. jannen","sally albrecht","velma"]
```

a. What might be a *generalizable* approach that will help us retrieve any first, middle, and last names (if they exist), for *all* the string examples in `name_list`?

FYI: A ***helper function*** is a function that ***encapsulates*** a smaller part of a larger problem we're trying to solve with another task (often, another function). When designing an algorithm, we ***decompose*** that algorithm into smaller pieces

b. Write out code for this ***helper function*** below:

```
def ______________________(a_string):
```

c. Observe the following code below. How does it differ from your solution?:

Sample Code

```
def split(a_string, char):
    """ splits a string into a list, based on given char
    >>>  split("oh hi doggie", ' ')
    ['oh', 'hi', 'doggie']
    """
    result = []
    curr_string = ''
    for ch in a_string:
        if ch == char:
            result = result + [curr_string]
            curr_string = ''
        else:
            curr_string = curr_string + ch
    result = result + [curr_string]
    return result
```

d. Write a line of code that uses this `split` function to grab the last name from the string `"artie q. jannen"`:

e. Write out Python code for a function, `last_names(name_list)` that takes a list of names and returns a list of just the last names in each name. Use the helper function, `split()`!

Application Questions: Use the Python Interpreter to check your work

1. Convert all your pseudocode in this activity to Python, and test it with a Python interpreter! Be sure to write good **docstrings** and **doctests**. Create doctests that will *stress test* your code using edge cases like empty string, empty list, and others!

2. Write a function, `most_vowels(name_list)` that takes a list of strings and returns a list of the names with the *most* number of vowels. You may find it helpful to write two **helper functions**: `is_vowel(char)` and `count_vowels(a_string).`

3. Write a function, `least_vowels(name_list)` that takes a list of strings and returns a list of the names with the *least* number of vowels. You may find it helpful to *reuse* your two helper functions: `is_vowel(char)` and `count_vowels(a_string).`