Name:_____ **Partner:** _____

Python Activity 23: Sets

An unordered sequence of unique items enables us to do some operations very efficiently!

Learning Objectives
Students will be able to:
Content:
- Define a **set**
- List example acceptable *types* for elements in a set
- Explain how **order**, **uniqueness**, and **mutability** apply to sets

Process:
- Write code that creates a set
- Write code to iterate over sets
- Write code that uses set theory methods to manipulate sets

Prior Knowledge
- Python concepts: mutability, lists, indexing, operators, for..loops, len(..), in, types

Critical Thinking Questions:

1. Examine the sample code in interactive python below.

Sample Code

```
0 >>> flwrs = {"rose", "daisy", "violet", "rose"}
1 >>> flwrs
```

a. Circle concepts new to us in this sample code.
b. How many elements does `flwrs` contain on line 0? _____
c. What do you think will be returned by the code on line 1?

_____

d. What is actually returned is `{'violet', 'daisy', 'rose'}`. How is this output different from what you expected in part (c)?

_____

e. What does the output in part (d) suggest about the *uniqueness* of elements in this new data structure? *(Hint: How many times does the element "rose" appear in line 0?)*

_____

f. What does the output in part (d) suggest about the *ordering* of elements in this new data structure?

_____

2. Examine the following code which continues to use this new data structure:

```
0 >>> fl_colors = {["rose", "red"], ["daisy", "white"]}
1 TypeError: unhashable type: 'list'
```

a. What is the *type* of elements in `fl_colors`? _____
How does it compare to the type of elements in `flwrs` in Quesiton 1? _____
b. Is the data structure of the elements of `fl_colors` *mutable* or *immutable*? _____
c. How might your response to (b) relate to the error thrown on line 1?

_____

d. What might this say about the *mutability* of <u>elements</u> in this new data structure?

_____

3. Examine the following code which continues to use this new data structure:

```
0 >>> flwrs = {"rose", "daisy", "violet", "rose"}
1 >>> flwrs[1]
```

a. What might be returned by the statement on line 1?

_____

b. What is output after line 1 is `TypeError: 'set' object is not subscriptable`. What might this error suggest about using numerical indices to access elements in this new data structure?

_____

FYI: Sets are *mutable, unordered* ***collections*** of unique, *immutable* objects.

c. What *type* of object might `flwrs` be? _____

4. If you had to guess, what do you think each of these set operators and functions do? (i.e., what might the code on the left do?)

Operator / Function	What the function/operator does
`len(my_set)`	
`empty_set = set()`	
`my_list = list(my_set)`	
`my_set = set(my_list)`	
`"rose" not in my_set`	
`for element in my_set:`	

5. Examine the sample code in interactive python below.

a. Why might line 0 (below) not throw an error, and line 2 does?

_____

```
0  >>> set([2, 0, 23])
1  {0, 2, 23}
2  >>> {[2, 0, 23]}
3  TypeError: unhashable type: 'list'
```

b. What additional python tests might we run to determine why line 0 (below) is False? (*Hint: Are we sure that empty curly brackets are a set?*)

_____

_____

```
0  >>> set() == {}
1  False
```

6. Circle the set operation on the right that describes what's happening with the code and its output on the left:

```
banana = {"yellow", "sweet", "fruit"}
lemon = {"fruit", "sour", "yellow"}
```

Code	Operation Description		
a. `>>> banana | lemon`	Union	Intersection	Difference
`{'fruit', 'sweet', 'sour', 'yellow'}`			
b. `>>> lemon & banana`	Union	Intersection	Difference
`{'fruit', 'yellow'}`			
c. `>>> banana - lemon`	Union	Intersection	Difference
`{'sweet'}`			
d. `>>> lemon - banana`	Union	Intersection	Difference
`{'sour'}`			

6. Examine the following code in interactive python:

```
0 >>> banana = {"yellow", "sweet", "fruit"}
1 >>> lemon = {"fruit", "sour", "yellow"}
2 >>> lemon |= banana
3 >>> lemon
4 {'sour', 'fruit', 'sweet', 'yellow'}
```

a. Circle the new operator in this code.

It's actually 2 operators we've seen before! What are they? _____ and _____

b. What might be the output of `lemon | banana`? _____

What might be the value of `lemon` after you execute `lemon | banana`? (*Hint: 5a)*

_____

c. What is the value of `lemon` after the code above is executed?

_____

d. What might the difference between your responses in (b) and (c) indicate about the *mutability* of sets?

_____

FYI: Sets, in Python, are essentially mathematical sets and support operations of mathematical **set theory** like union (**|**), intersection (**&**), and difference (**-**). If we combine those operators with an assignment operator, they will overwrite the variable on the left-hand side of of operator (i.e., **`s1 |= s2`** is the same as **`s1 = s1 | s2`**).

Application Questions: Use the Python Interpreter to check your work

1. Write a function, `is_one_row(word)`, that takes a string, `word`, as an argument and returns True if and only if the given word can be typed all in one single row of keyboard key rows (`qwertyuiop, asdfghjkl, zxcvbnm`). The word "type" is an example of this. (*Hint: use sets!*)

```
def is_one_row(word):
```

2. Write a function, `is_isogram(word)`, that takes a string, `word`, as an argument and returns True if that word contains letters that appear only once. (*Hint: use sets!*)

```
def is_isogram(word):
```

3. Write a function, `is_subset(word, hive)`, that takes two string arguments, `word` and `hive`, and returns True if all the letters in `word` appear in `hive`. (*Hint: use sets!*)

```
def is_subset(word, hive):
```

4. Write a function, `spelling_game(required, hive, word_list)`, that takes 3 string arguments, and returns a list of all words in `word_list` that are (1) at least 4 letters long, (2) use the letter `required`, (3) only have letters that appear once, and (4) have only letters that appear in `hive`. (*Hint: use your functions is_isogram and is_subset!*)

```
def spelling_game(required, hive, word_list):
```

Note: spelling_game is essentially the NYTimes Spelling Bee Game: https://www.nytimes.com/puzzles/spelling-bee