Name:______________________________ **Partner:** __________________________

Python Activity 25b: Classes - Methods

Learning Objectives
Students will be able to:
Content:
- Define **methods** and **initializers** in python
- Identify differences between methods/functions and attributes/variables

Process:
- Write code that creates a new user-defined class with methods and initializers

Prior Knowledge
- Python concepts from Activities 1-24.

Folks, this is a brand new activity. If you encounter any issues/typos, please let Iris know!

Critical Thinking Questions:

1. Examine the following code from interactive python below.

Interactive Python

```
>>> example = list()
>>> example.append(2)
>>> example.append(4)
>>> example
[2, 4]
```

a. What type of object is `example`? How do you know?

____

b. When we call `.append()` which object are we appending to? How do you know?

____

c. If we reassigned `example` to be `'24'` what would `.append()` do?

____

FYI: Functions that operate on certain kinds of objects are called **methods** (`.append()` is a method of List). We have been using many methods since the beginning of the course.

d. What are some additional methods that we have been using in this course so far?

For lists: ____________________________________

For strings: ___________________________________

2. Examine the following code below, that creates a new class in interactive python:

```
0 >>> class EvensList:
1 ...      """ A new class to store data """

2 >>> el = EvensList()
3 >>> el.items = [2,4]
4 >>> el.items
5 [2, 4]
6 >>> el.append(6)
```

a. What type of object is `el`? How do you know?

b. What value does `el.items` hold after line 3?

c. What type of object is `el.items`? How do you know?

d. What attributes does `EvensList` have?

e. What does the programmer hope will happen after line 6?

f. This code will generate the following error, "`AttributeError: 'EvensList' object has no attribute 'append',`" why do you think that is?

3. Observe what happens when we enter the following lines, continuing from those above:

```
8  >>> def append(evenlst, item):
9  ...     evenlst.items.append(item)

10 >>> append(el, 6)
11 el.items
12 [2, 4, 6]
```

a. How does line 10 in this example differ from line 1 in question 1?

b. Is `append(..)` defined on lines 8 & 9 a method or a function? Why?

FYI: User-defined object instances can be passed to functions just like built-in object instances.

c. How does the value of `el.items` change in line 10?

FYI: User-defined object instances are mutable.

d. Write some lines of python to adjust the append function so that it only adds items to `evenlst` that are even numbers:

   `def append(evenlst, item):`

4. Examine the following code below, that creates a new class in interactive python:

```
0 >>> class EvensList:
1 ...     def append(self, item):
2 ...         self.items.append(item)

4 >>> el = EvensList()
5 >>> el.items = [6,4]
6 >>> el.append(3)
7 >>> el.items
8 [6, 4, 3]
```

a. What value does `el.items` hold after line 6?

b. How does the call to `append` differ in line 6 in this example, versus line 10 in question 3?

c. How does `append's` function header differ in line 1 above versus line 8 in question 3?

d. How does `append's` function definition differ in line 2 above versus line 9 in question 3?

FYI: In user-defined types, we refer to the values stored in that instance through the keyword, **self**.

e. If we were to add a line 3 to the `append` method that was `print(self.items)` what might be printed and on after what line?

f. Modify the append method for `EvensList` to only append integers that are even numbers:

5. Examine the following code below, that creates a different version of `EvensList`, but as a script:

EvensList.py

```
0 class EvensList:
1    def __init__(self, itemList):
2         self._items = itemList
3    def append(self, item):
4         self._items.append(item)

5 if __name__ == '__main__':
6    betterEL = EvensList([88, 12, 4])
7    print(betterEL._items)
8    # prints [88, 12, 4]
9    betterEL.append(8)
10   print(betterEL._items)
```

a. What two lines did we add to this definition of `EvensList` that we did not see in the previous question?

b. How does our creation of the `betterEL` variable on line 6 differ in this example from creating `el` in the previous example?

FYI: The **__init__** method is *implicitly* called when you instantiate a new object. It is very useful for setting up an object with an initial state or initial values.

c. What's stored in `betterEL._items` when line 7 is printed?

d. What's stored in `betterEL._items` after line 9 is executed?

Application Questions: Use Python to check your work

1a. Create a class, **`Book`**, which has a **`string`** as slot. Write a method `printText` that will print to the screen the text that the book contains.

```
class Book():
    __slots__ = ['theText']
    def printText(self):
```

1b. Write a method for `Book`, `readSome`, that prints to the screen the first eighty characters from the text.

```
    def readSome(self):
```

1c. Add an attribute to `Book` that keeps track of the index of the last character read by `readSome()`. Update `readSome()` to change that value when it reads from the text, and to only begin reading from where it last left off:

1d. Write an initializer for the `Book` class that takes an initial text and stores it as an attribute/slot:

```
def __init__(self, txt):
```

1e. Write a main function for the Book class that creates a new book with text and uses the methods you wrote in the previous questions:

```
if __name__ == '__main__':
```