Name:_____ **Partner:** _____

Python Activity 36: Classes – Accessors & Mutators

Digging deeper into the useful aspects of user-defined types with attributes and methods.

Learning Objectives

Students will be able to:

Content:

- Describe what a variable name with leading underscore implies
- Explain the difference between **public**, **protected**, and **private** access
- List differences between **accessor** and **mutator** methods

Process:

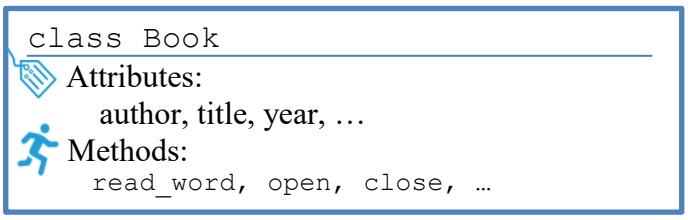
- Write code that creates a new user-defined class with initializer method
- Write code that creates a new user-defined class with accessor & mutator methods

Prior Knowledge

- Python concepts: user-defined classes, methods, attributes, class object model, self

Concept Model:

Recall the potential *Class Object Model* (below) for the `Book` class from the example "Iris reads J.R.R. Tolkein's *The Fellowship of the Ring,* originally published in 1954.":



CM1. What are the *attribute values* for this example?

_____

_____

Critical Thinking Questions:

1. Examine the following code below.

book.py

```
class Book:
    def __init__(self, book_author, book_title, book_year):
        self._author = book_author
        self._title = book_title
        self._year = book_year
```

a. What is new about the `Book` class's attribute variables that we haven't seen before?

_____

FYI: In Python there is an ***attribute naming convention*** that indicates that variable names that start with a single leading underscore (_) *should* not be accessed from outside the class in which they're defined. We call these ***protected*** variables. In Python, these are conventions, not rules, but we will

follow them. ***Private*** variables are indicated with a leading double underscore, and can *only* be accessed within the class, python enforces this.

b. For each of the potential attribute names below, circle if they are public, protected, or private:

Attribute Name:	*Circle one:*		
`copyright`	public	protected	private
`_address`	public	protected	private
`__edition`	public	protected	private

c. For each OOP situation on the left, circle one of the access-terms on the right:

OOP Attribute Situation	*Circle one*
If the attribute *should* only be accessed (modified, used, etc.) from within the class itself and its subclasses.	public protected private
If the attribute *can* only be accessed (modified, used, etc.) from within the class itself.	public protected private
If the attribute *can* and *should* also be accessed (modified, used, etc.) from outside the class, as well as within.	public protected private

d. Why might most attributes in our CS134 class start with a single underscore?

_____

2. Examine the following code below, that extends our previous code:

book.py

```
class Book:
    """This class represents a book """
    def __init__(self, book_title, book_author, book_year):
        self._title = book_title
        self._author = book_author
        self._year = book_year

    def get_title(self):
        return self._title

if __name__ == "__main__":
    lotr = Book("Fellowship of the Ring", "Tolkein", 1954)
    print(lotr.get_title())
```

a. Place a star next to the code that is new in this example.

b. The last line, prints `Fellowship of the Ring`. Why might that be?

_____

FYI: ***Accessor*** methods retrieve the values of private and protected attributes from outside of the class definition.

c. Write two lines of code to add an additional *accessor method* to our `Book` class, to get the value of the `Book` instance's year of publication:

d. Write a line of code that uses this new *accessor method* of our `Book` class from (c):

3. Examine the following code below, that extends our previous code:

book.py

```
class Book:
    """ This class represents a book """
    def __init__(self, book_title, book_author, book_year):
        self._title = book_title
        self._author = book_author
        self._year = book_year

    def get_title(self):
        return self._title

    def set_title(self, book_title):
        self._title = book_title

if __name__ == "__main__":
    lotr = Book("Fellowship of the Ring", "Tolkein", 1954)
    lotr.set_title("Book One")
    print(lotr.get_title())
```

a. Place a star next to the code that is new in this example.

b. When we call `lotr.set_title(..)` just before the last line of code, what might be happening to the `lotr` instance's attribute values?

c. What might be printed by `lotr.get_title()` on the last line? _____

FYI: ***Mutator*** methods *set or change* the values of the attributes, when outside of the class implementation.

d. Write two lines of code to add an additional *mutator method* to our `Book` class, to set the value of the `Book` instance's year of publication:

e. Write a line of code to use this *mutator method* of our `Book` class from (d):

4. Examine the following code below, that extends our previous code:

book.py

```
0  class Book:
1      """ This class represents a book """
2      def __init__(self, book_title, book_author, book_year):
3          self._title = book_title
4          self._author = book_author
5          self._year = book_year

6      def get_author(self):
7          return self._author

8      def same_author_as(self, other_book):
9          return other_book.get_author() == self._author

10 if __name__ == "__main__":
11     lotr = Book("Fellowship of the Ring", "Tolkein", 1954)
12     pp = Book("Pride & Prejudice", "Austen", 1813)
13     emma = Book("Emma", "Austen", 1815)
14     print(lotr.same_author_as(pp))
15     print(emma.same_author_as(pp))
```

a. Place a star next to the code concepts that are new to us in this example.

b. What would be the output of the following commands:

`lotr.get_author()` _____

`pp.get_author()` _____

`emma.get_author()` _____

c. What arguments does the `same_author_as(..)` method require?

_____ and _____

d. When `lotr.same_author_as(pp)` is called on line 14, how do the *parameter values* from the function call match to the arguments of the function definition?

The `self` argument is replaced with the _____ object.

The `other_book` argument is replaced with the _____ parameter value.

e. According to your response in (d) what is returned by the following values when line 14 is executed? `self._author` _____

`other_book.get_author()` _____

f. What might be printed by the call to `lotr.same_author_as(pp)`? _____

g. What might be printed by the call to `emma.same_author_as(pp)`? _____

h. Create a new method, `num_words_in_title()`, which returns the number of words in the title of the book:

Application Questions: Use Python to check your work

1. Continue implementing our class, `Book`:

a. Add a method for `Book`, `years_since_pub(current_year)`, that takes in the current year and returns the number of years since the book was published (*Hint: Don't forget `self`!*):

b. Create two different instances of `Book` objects:

c. Write some lines of code that use the methods you wrote on the `Book` instance objects: