

Name: _____

Partner: _____

Python Activity 35: Introduction to Classes

We can use concepts such as abstraction and encapsulation to create our own types!

Learning Objectives

Students will be able to:

Content:

- Define **instances**, **objects**, **methods**, **attributes**, and **classes** in python
- Identify differences between methods and functions
- Describe when to include **self** as a parameter/argument, and when not to

Process:

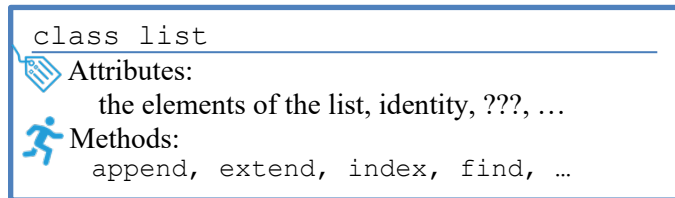
- Write code that creates a new user-defined class with methods

Prior Knowledge

- Python concepts: lists, types, methods, dot notation

Concept Model:

CM1. Examine the following *class object model* for the `list` class in Python:




a. If we create a new list object, with the line: `names = ["iris", "lida", "mark"]`, what are some of the *methods* we can use on `names`?

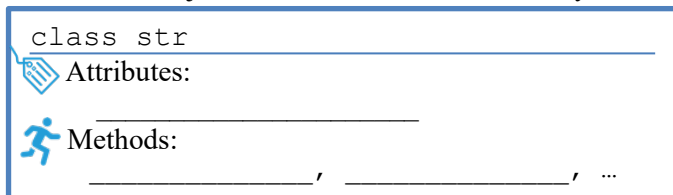
b. What might be the *attribute* of `names`? _____

FYI: *Abstraction* is the hiding of the details of implementation. As an example, we've been using `lists` all semester without knowing how the methods are implemented or the exact *data representation* of the attributes.

c. If `names` is an *instance* of a list class object, how might we write a line of code to create another *instance* of a list object? _____

 d. How might we define what an *instance of a class* is? _____

CM2. a. Complete this class object model for the `str` class in Python:



 b. What might be the difference between *attributes* and *methods*? _____

- CM3. In Python, we can create our own *data types* (or *classes*). In the following questions, consider the important features of a *book*. As an example:

Iris reads J.R.R. Tolkein's *The Fellowship of the Ring*, originally published in 1954.

- What are the *attributes* of this book? _____, _____, _____
- What might be a *method* we can perform on/with books in the example? _____
- In this example, what might be the *class* and what might be the *instance* of the class?

Class: _____ Instance: _____

Critical Thinking Questions:

- Examine the following code from interactive python below.

```
Interactive Python
0 >>> example = list()
1 >>> example.extend([2, 4])
2 >>> example
3 [2, 4]
```

- What *type* of object is example? How do you know?

- Fill in the blank: example is an *instance* of a _____ object.
- When we call `.extend()` which object are we extending? How do you know?

- If we reassigned example to be "24" what would `.extend()` do? Why?

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

```
0 >>> class SampleClass:
1 ...     """Class to test the use of methods """
2 ...     def greeting(self):
3 ...         print("Hello")

4 >>> sample = SampleClass()
5 >>> sample.greeting()
6 Hello
```

- What type of object is sample? How do you know? (*Hint: Refer to question 1a*)


- Fill in the blank: sample is an *instance* of a _____ object.
- Which lines are indented *under* class SampleClass? _____

- d. When we call `.greeting()` on line 5 which object are we calling it on? How do you know? _____
- e. If we reassigned `sample` to be "24" what might `sample.greeting()` do? How do you know? _____

3. Examine the following code below which is similar to our previous example:

```
0 >>> class SampleClass:
1 ...     """Class to test the use of methods """
2 ...     def greeting(self):
3 ...         print("identity:", id(self))


4 >>> sample = SampleClass()
5 >>> sample.greeting()
6 identity: 439025
7 >>> id(sample)
8 439025
```

- a. Underline the code that is different in this example.
- b. How do the identities of `self` and `sample` compare? _____
-  What does this imply about `self` and `sample`? Is `self is sample` True or False? _____
- c. What might the argument `self` refer to? _____

FYI: To create methods that can be called on an instance of a class, they must have a parameter which takes the instance of the class as an argument. In Python, the ***first parameter of a method is always `self`, and is used as a reference to the calling instance***. All methods include `self` as the first parameter!

When ***defining methods***, always include `self`

When ***calling methods***, the value for `self` is passed implicitly (i.e., we don't specify it, but it happens automatically).

-  d. Why is `self` not passed as an argument on line 5? _____

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

```
0 >>> class SampleClass:
1 >>>     """Class to test the use of methods """
2 >>>     def __init__(self):
3 >>>         print("__init__ is called")

4 >>> sample = SampleClass()
5 __init__ is called
```

- a. Fill in the blank: `sample` is an *instance* of a _____ object.

b. Circle the two places where we see `__init__` is called.

c. Circle the `__init__` method call. (*Hint: Trick question!*)



d. What must be happening on line 4, to produce the output we see on line 5?

Application Questions: Use Python to check your work

1a. Create a class, `Book`, from Concept Model #3, which has an *initializer* method that will print "Creating a new book" when a new *instance* of `Book` is created:

1b. Add a method, `open`, that will print "The book is open." to the display when called:

1b. Add a method for `Book`, `close`, that prints to the screen "Blam!".

1c. Write a line of code to create a new instance of a `Book`, object:

1d. Write some lines of code that use the methods you wrote on the `Book` instance object:

*The next POGIL will introduce defining attributes for new classes,
which lets us start to build interesting new data types!*