Prelab 12: Classes in Python

What are objects/classes?

Objects are way of wrapping (or encapsulating) values so they can be referenced as single value (for more on encapsulation, see https://www.google.com/search?q=encapsulation). In Python, we create objects by by writing “Classes”.

For example, if we would like to track the information related to several college courses, we might define a class called Course.

```python
class Course:
    # Member variables
    faculty = ""  # String - name of the course instructor
    department = ""  # String - department hosting the course (ie: 'CS')
    number = 0  # Integer - course number (ie: 177)
    title = ""  # String - course title (ie: 'Object Oriented Programming')
    size = 0  # Integer - number of students enrolled

    def show(self):
        # Prints the course details to the console
        print("Course    : ", self.department, self.number)
        print("Faculty   : ", self.faculty)
        print("Title     : ", self.title)
        print("Size      : ", self.size)
```

The above class definition creates the Course class which includes the member variables faculty, department, number, title, size and it sets their default values. When a function is defined within the class it is called a Method. In this case, the show method accepts the argument “self” and then prints the member variables from the course object.

How do we create course objects using the "Class" object definition?

In the Course definition, number, department, etc.. are called member variables - they are values that belong to the class Course and have been *encapsulated* together to form the class Course. Once a class is defined, instances of the class can be created by creating Objects - similar to examples in Graphics for Point, Window and GraphWin. Accessing the variables defined within the class as well as the methods defined can be done by referring to them along side with the object.

```python
>>> cs177 = Course()
```

At this point, we've created an object named cs177 of type Course and it's member variables have
been defined with default values.

The function `show()` takes an argument 'self' which refers to the object that is calling this particular function. In the code above, the `show` function was called that belong to the `Object cs177`. The variables within 'show()' refer to the member variables of the object that called the function.

```python
>>> cs177.show()

Course   :   0
Faculty   :
Title     :
Size      :  0
```

Usage of getters/setters as a best practice

One of the best practices of programming Classes are by the way of defining getters/setters which are methods used to access and set member variables. In the class definition below, the department, number and faculty member variables are used to demonstrate to the way to implement the getter/setter.

Examples of Setters

```python
def setFaculty(self, faculty):
    # Modifies the faculty based on the given argument
    # ie:  myCourse.setFaculty('Thomas, Jonathon')
    self.faculty = faculty

def setDepartment(self, department):
    # Modifies the course title based on the given argument
    # ie:  myCourse.setDepartment('CS')
    self.department = department

def setNumber(self, number):
    # Modifies the number of students enrolled based on the given argument
    # ie:  myCourse.setNumber(101)
    self.number = number
```

```python
>>> cs177 = Course()
>>> cs177.setFaculty('Rego, Vernon')
>>> cs177.setDepartment('CS')
>>> cs177.setNumber(177)
>>> cs177.setSize(200)
>>> cs177.show()

Course   :   CS 177
Faculty   :  Rego, Vernon
Title     :
Size      :  200
```
Examples of Getters

Getters are used in order to access member variables. Each member variable provides with a method which returns itself to the caller. This method is better than directly accessing the variables as seen above because it is more resilient to changes in member variables.

```python
def getFaculty(self):
    # Returns the faculty teaching the course
    # ie: myCourse.getFaculty()
    return self.faculty

def getDepartment(self):
    # Returns the department of the course
    # ie: myCourse.getDepartment()
    return self.department

def getNumber(self):
    # Returns the number of students enrolled
    # ie: myCourse.getNumber()
    return self.number
```

```python
>>> cs177.getFaculty()
'Rego, Vernon'
>>> cs177.getDepartment()
'CS'
```

Constructors - methods for creating objects

The `__init__` method (referred to as the constructor method) is used for creating objects and simultaneously defining values for the member variables. This allows the programmer to create an object and set the member variables in one step, instead of requiring them to set them one at a time using the setter methods.

```python
class Course:
    # Member variables
    faculty = ''  # String - name of the course instructor
    department = ''  # String - department hosting the course (ie: 'CS')
    number = 0  # Integer - course number (ie: 177)
    title = ''  # String - course title (ie: 'Object Oriented Programming')
    size = 0  # Integer - number of students enrolled

    # Constructors
    def __init__(self, course_faculty, course_department, course_number, course_title, course_size):
```

Computer Science Courses - http://courses.cs.purdue.edu/
# __init__ creates an object-instance of the _type_ Course. 
# It accepts as arguments the faculty, department, number, title 
# and size and assigns the member variables to the input. 
# (ie: >>> c = Course("CS", 17700, "Python Programming", 200)

```python
self.faculty = course_faculty
self.department = course_department
self.number = course_number
self.title = course_title
self.size = course_size
```

```python
>>> cs182 = Course('Dunsmore, H', 'CS', 182, 'Intro to Java', 215)
>>> cs182.show()
```

<table>
<thead>
<tr>
<th>Course</th>
<th>CS 182</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faculty</td>
<td>Dunsmore, H</td>
</tr>
<tr>
<td>Title</td>
<td>Intro to Java</td>
</tr>
<tr>
<td>Size</td>
<td>215</td>
</tr>
</tbody>
</table>

**Review of Object Definitions**

To review the concept of creating objects using classes in Python -

1. Defining a class with variables and methods
2. Creating an object of the class, and ability to access/modify the using getters/setters for member variables
3. Defining constructors for creating objects with initial member variables.

**Tasks to complete before lab**

Download the Course.py file here and complete complete the unfinished methods following the examples shown in the Course class definition. There are four (4) including:

- setTitle
- setSize
- getTitle
- getSize

You will be using the Course class definition in Lab 12, so it's important that you complete this Prelab assignment prior to your lab meeting.