Classes (*Liang* Ch. 7, pp 216-237)

- Classes are convenient wrappers that allow us to gather related data elements into one location. Classes provide two important things:
  - A group of related *properties* (or data fields) that we can set or get.
  - A set of *methods* (similar to functions) that perform useful operations on the *properties*.
- Objects are *instances* of a class. Strings and Lists are actually classes and `sort` is one of the *methods* of the List class. For example:

```
L = [3,2,1]  # L (the object) is an instance of the List class
L.sort()
```

- Classes are declared using the keyword *class* and the name of the class. By convention, class names are capitalized:

```
class Midshipman:
```

Example

```
class Midshipman:
    alpha = 211234
    company = 13

mid1 = Midshipman()
mid2 = Midshipman()

print(mid1.alpha,mid1.company)
print(mid2.alpha,mid2.company)
```

Every Instance Of A Class Has The Same Values?

- Classes would have limited utility if every instance of a given class had the same values in its properties.
- We need a way to initialize new instances of a given class with specific values. That's where *initializers* come in. Initializers are Class methods and are always named: `__init__`
- They look like this:

```
class Midshipman:
  def __init__(self,alphaCode,companyNumber):
    self.alpha = alphaCode
    self.company = companyNumber
```

self?

- The name *self* has special meaning in python (and many languages).
- The first argument of every class method, including `__init__`, is always a reference to the current instance of the class. By convention, this argument is always named *self*. You don't need to pass *self* to a class method when you invoke it, but you do need to include it in any method definitions for the class.
- Initializers often use the same names for parameters and properties, and *self* allows us to tell them apart. Our *Midshipman* Class initializer could be written as:

```
class Midshipman:

   def __init__ (self,alpha,company):
      self.alpha = alpha
      self.company = company
```

Example

```
class Midshipman:
  def __init__ (self,alpha,company):
    self.alpha = alpha
    self.company = company

mid1 = Midshipman(211234,13)
mid2 = Midshipman(219999,8)

print(mid1.alpha,mid1.company)
print(mid2.alpha,mid2.company)
```

Add Some More Methods

- Class methods can operate on the properties of an object without having to pass them in as parameters.

```
class Midshipman:

   def __init__ (self,alpha,company):
      self.alpha = alpha
      self.company = company

   def regiment(self):
      if self.company <= 15:
         reg = "First"
      else:
         reg = "Second"
      return reg

mid1 = Midshipman(211234,13)
print(mid1.regiment())
mid2 = Midshipman(219999,22)
print(mid2.regiment())
```

Class Methods Can Also Take Arguments

```
class Midshipman:

  def __init__ (self,alpha,company,prtScore):
    self.alpha = alpha
    self.company = company
    self.prtScore = prtScore

  def prtBonus(self,additionalPoints):
    self.prtScore += additionalPoints

mid1 = Midshipman(211234,13,81)
print(mid1.prtScore)
mid1.prtBonus(7)
print(mid1.prtScore)
```

Properties Can Be Any Legal Python Type

```
class Midshipman:
   def __init__ (self,alpha,prtScore):
      self.alpha = alpha
      self.prtScore = prtScore

class Company:
   def __init__ (self):
      self.mids = []

def addMid (self,mid):
   self.mids.append(mid)

def printScores (self):
   for mid in self.mids:
      print(mid.alpha,mid.prtScore)
```

```
# New empty Company
co13 = Company()

# Create and add mids
mid = Midshipman(211234,81)
co13.addMid(mid)
mid = Midshipman(218888,93)
co13.addMid(mid)
mid = Midshipman(219999,88)
co13.addMid(mid)

# Print prt scores for all mids
co13.printScores()
```

Properties Can Even Be Objects Of Other Classes

Your Turn

Class name: CyberStudent

Properties: alpha : *string*
hwScores : *[int]*

Methods:

1. *__init__(self,alpha)* Initializes a new CyberStudent with *alpha*, and sets the *hwScores* property to an empty list.
   Returns: Nothing

2. *addHW(self,score)* Appends *score* to the *hwScores* property.
   Returns: Nothing

3. *average(self)* Computes the average of all the scores in the *hwScores* property.
   Returns: hwAverage : *float*

- Design the *CyberStudent* class, including both properties and all three methods.
- Write a short program that creates a new *CyberStudent* object.
- Add three homework grades of your choosing (integers) to her record.
  - Note: you should have designed *average* to compute the average for any length *hwScores* list, not just a list of length three.
- Have your program print the average of her three homework grades using the *average* method for the *CyberStudent* class.