CMSC201
Computer Science I for Majors
Lecture 17 – Classes and Modules ( Continued )

Prof. Katherine Gibson
Last Class We Covered

• More about “good quality” code
• Modules
• The `import` keyword
  – Three different ways to import modules
• Classes
  – Creating an instance of a class
  – Vocabulary related to classes
Any Questions from Last Time?
Today’s Objectives

• To review the vocabulary for classes
• To better understand how constructors work
• To learn the difference between
  – Data attributes
  – Class attributes
• To explore special built-in methods and attributes
Class Vocabulary

class student:

def __init__(self, name, age):
    self.full_name = name
    self.age = age

def get_age(self):
    return self.age
class student:

def __init__(self, name, age):
    self.full_name = name
    self.age = age

def get_age(self):
    return self.age

class method

constructor

keyword

class name

current instance

class members (or attributes)
Creating Instances of a Class
Constructor

• In order to use a class we have created, we have to be able to create *instances* of it to use.

• We can accomplish this using a special type of method (*i.e.*, a class function) called a *constructor*.
  
  – Using it will allow us to “construct” instances of our class.
The constructor has a special name: the word “init” with two underscores in front of it, and two underscores in back

- This special name tells Python how to use it

The `__init__()` method needs to be contained inside our class

- It normally does initialization of the class data members and other important things
Constructor Example

• Here is an example constructor for `student`

```python
class student:
    def __init__(self, name, age, gpa):
        self.name = name
        self.age = age
        self.gpa = gpa
```

• It takes in three arguments (plus `self`) and initializes our data members with them
Using a Constructor

• To use our constructor:
  – Use the class name with () notation
  – Pass in the arguments it needs
  – Assign the results to a variable

\[
\text{test1} = \text{student("Jane", 22, 3.2)}
\]

• Creates a new \textbf{student} object called \texttt{test1}
Constructor Code Trace

• What happens when we call a constructor?

```python
def main():
    test1 = student("Jane", 22, 3.2)

    def __init__(self, name, age, gpa):
        self.name = name
        self.age = age
        self.gpa = gpa
```
Constructor Code Trace

• What happens when we call a constructor?

def main():
    test1 = student("Jane", 22, 3.2)

    name = "Jane"
    age = 22
    gpa = 3.2

    def __init__(self, name, age, gpa):
        self.name = name
        self.age = age
        self.gpa = gpa
Constructor Code Trace

• What happens when we call a constructor?

```python
def main():
    test1 = student("Jane", 22, 3.2)
```

Notice that all of the local variables in `__init__` disappeared!

```python
def __init__(self, name, age, gpa):
    self.name = name
    self.age = age
    self.gpa = gpa
```

Creates and returns a `student` object.
The `self` Variable

• The `self` variable is the first parameter of every single class method – we must use it!
  – But we **don’t** explicitly pass it in
  – Python **implicitly** passes it in (for us!)

• Calling the constructor:
  ```python
test1 = student("Jane", 22, 3.2)
```

• The constructor definition:
  ```python
def __init__(self, name, age, gpa):
```
The `self` Variable

- The `self` variable is how we refer to the current instance of the class.
- In `__init__`, `self` refers to the object that is currently being created.
- In other methods, `self` refers to the instance the method was called on.
Deleting an Instance

• Some languages expect you to delete instances of a class after you are done with them
  – Python is not one of those languages

• Python has automatic “garbage collection”
  – It automatically detects when all of the references to a piece of memory have gone out of scope
  – Generally works pretty well
Attributes
Attributes

• There are two types of attributes:

1. Data attributes
   – Also called instance variables

2. Class attributes
   – Also called class variables
Data Attributes

• **Data attributes**
  – Variables are owned by a particular instance
  – Each instance has its own value for each attribute

\[
\begin{align*}
test1 & = \text{student("Jane", 22, 3.2)} \\
\text{name: } & \text{"Jane"} \\
\text{age: } & 22 \\
\text{gpa: } & 3.2 \\
\end{align*}
\]

\[
\begin{align*}
test2 & = \text{student("Adam", 19, 1.9)} \\
\text{name: } & \text{"Adam"} \\
\text{age: } & 19 \\
\text{gpa: } & 1.9 \\
\end{align*}
\]
Data Attributes

• Data attributes are created and initialized by the class’s `__init__` method

• Inside the class, data attributes must have “self.” appended to the front of them

```python
def setAge(self, age):
    if age > 0:
        self.age = age
    else:
        self.age = 1
```
Class Attributes

• **Class attributes** are owned by the whole class

• All instances share the same value for it
  – When any instance of the class changes it, it changes for all instances of the class

• Class attributes are often used for:
  – Class-wide constants
  – Counting how many instances of a class exist
Class Attributes

• Class attributes must be defined within the class definition, but outside any methods

```python
class student:
    MAX_ID_LENGTH = 4  # constant
    numStudents = 0     # counter

    def __init__(self, name, age, gpa):
        # __init__ method definition...

    # rest of class definition
```
Class Attributes

• Since there is one of these attributes per class and not one per instance, they’re accessed via a different notation:

    self.__class__.name

    – Use the actual keyword “class”

    – This is the safest way to access these attributes

    def increment(self):
        self.__class__.numStudents += 1
Data vs. Class Attributes Example

class counter:
    # class attribute
    overall_total = 0

    def __init__(self):
        # data attribute
        self.my_total = 0

    def increment(self):
        self.my_total += 1
        self.__class__.overall_total += 1
Data vs. Class Attributes Example

```python
one = counter()
two = counter()
one.increment()
two.increment()
two.increment()

print("one's total", one.my_total)
print("class total", one.__class__.overall_total)
print("two's total", two.my_total)
print("class total", two.__class__.overall_total)
```

```
one's total 1
class total 3
two's total 2
class total 3
```
Special Built-In Methods
Built-In Methods

• Python automatically includes many methods that are available to every class
  – Even if you don’t explicitly define them

• These methods define functionality triggered by special operators or usage of that class

• All built-in methods have double underscores around their name: __init__
Special Methods

• Here are some special methods and their uses:

__init__
  – The constructor for the class
  – Often initializes the data members

__repr__
  – Defining how to “turn” an instance into a string
  – Used whenever we call `print()` with an instance
More Special Methods

• There are additional special methods, including ones that let you define how these work:
  – Comparison
  – Assignment
  – Copying
  – `len()`
  – Using `[]` notation like a list
  – Using `()` notation like a function
Special Built-In Attributes
Built-In Attributes

• Python also has special attributes that exist for all classes

___class___
  – Gives a reference to the class from any instance
  – We already use this for accessing class attributes

___module___
  – Gives a reference to the module it’s defined in
The __doc__ Attribute

• We can also use documentation strings in our class, and access them using __doc__

• To add documentation, use 3 double quotes

class student:
    """This is a class for a student""
    MAX_ID_LENGTH = 4
    numStudents = 0

    def __init__(self, name, age, gpa):
        """Constructor for a student""
        # constructor definition...
The __doc__ Attribute

• To access the documentation, use __doc__

test1 = student("Jane", 22, 3.2)

print(test1.__doc__)
print(test1.__init__.__doc__)

This is a class for a student
Constructor for a student
The `dir()` Function

• If you want a list of all the available attributes and methods, you can call the `dir()` function on any instance of the class:

```python
dir(testStudent)
```

```python
['MAX_ID_LENGTH', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'age', 'checkGraduate', 'getNumStudents', 'gpa', 'idNum', 'increment', 'name', 'numStudents', 'printStudent', 'setAge', 'setIDNum']
```
If we have time...

LIVECODING!!!
Any Other Questions?
Announcements

• Midterm Survey (on Blackboard)
  – Due by Friday, November 6th at 8:59:59 PM

• Project 1 is out
  – Due by Tuesday, November 17th at 8:59:59 PM
  – Do NOT procrastinate!

• Next Class: Inheritance