Lecture 15:
Classes
(Chapters 15 & 17.1-17.5)

CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Announcements

- Prelim 2 alternate time request form live Fri 3/25
- More 1-on-1’s today thru Sunday.
  - Come one, come all (Sign up on CMS.)
- A5 due date moved later to Sun 4/17.
  - The tradeoff: more time to work on A5, less “pressure” on Spring break, BUT less time to look at the A5 solutions before Prelim 2 (Tu 4/19) and temptation to delay prelim studying. (Resist that temptation.)
- next week’s lab 16 extended to Wed 4/13 due to spring break
- These updates are on the Schedule webpage.

Recall: Objects as Data in Folders

- attributes: variables within objects
- Type shown in the corner
- unique identifier

nums = [2,3,5]
nums[1] = 7

Classes are user-defined Types

Defining new classes = adding new types to Python

class name

class name

Example Classes

- Point3
- Rect
- Freq (A3), for word frequencies
- Doll (class, lab)
- Song, Mix (A4)

Simple Class Definition

```python
class <class-name>:
    """Class specification""
    <method definitions>
```

Just like function definitions, but placed inside a class definition, i.e., indented relative to the class header
The Class Specification

class Course:
    """An instance is a Cornell course
    Instance Attributes:
    name: [str] name of the course of form: <DEPT NUM>
    n_credit: [int] number of credits, must be > 0
    """

Convention: capitalize first letter of class name

Attribute list
Short Summary
Description and invariant*

*more about this later in this lecture

Constructor (1)

- Function to create new instances
  - function name is the class name
- Calling the constructor:
  - Makes a new object (folder) on the Heap
  - Returns the id of the folder

But how do we populate the folders?

c1 = Course("CS 1110", 4)
c2 = Course("MATH 1920", 3)

Heap Space
id1
Global Space
c1
id1
id2
c2

Constructor (2)

- Function to create new instances
  - function name is the class name
- Calling the constructor:
  - Makes a new object (folder) on the Heap
  - Calls the __init__ method
  - Returns the id of the folder

Special Method: __init__

def __init__(self, name, n_credit):
    """Initializer: creates a Course
    name:     [str] name of the course
    n_credit: [int] num credits, must be > 0
    """
    self.name = name
    self.n_credit = n_credit
    # this is the call to the constructor, which calls __init__

Heap Space
id1
Global Space
id1
id2
course

Evaluating a Constructor Expression

1. Constructor creates a new object (folder) of the class Course on the Heap
   - Folder is initially empty
   - Has id
2. Constructor calls __init__ (self, "CS 1110", 4)
   - self = identifier ("Fill this folder!")
   - Other args come from the constructor call
   - commands in __init__ populate folder
   - __init__ has no return value ("I filled it!")
3. Constructor returns the id
4. LHS variable created, id is value in the box

Truths about Object Instantiation

1) Instantiate an object by calling the constructor
2) The constructor creates the folder
3) A constructor calls the __init__ method
4) __init__ puts attributes in the folder
5) The constructor returns the id of the folder
### Invariants

- Properties of an attribute that must be true
- Works like a precondition:
  - If invariant satisfied, object works properly
  - If not satisfied, object is “corrupted”
- **Example**:
  - `Course` class: attribute `name` must be a string
- Purpose of the **class specification**

### Checking Invariants with an `Assert`

```python
class Course:
    """Instance is a Cornell course """
    def __init__(self, name, n_credit):
        """Initializer: instance with name, n_credit courses"
        name: [str] name of the course of form: <DEPT NUM>
        n_credit: [int] num credits, must be > 0
        """
        assert type(name) == str, "name should be type str"
        assert name[0].isalpha(), "name should begin with a letter"
        assert name[-1].isdigit(), "name should end with an int"
        assert type(n_credit) == int, "n_credit should be type int"
        assert n_credit > 0, "n_credit should be > 0"
        self.name = name
        self.n_credit = n_credit
```

### Student Class Specification, v1

```python
class Student:
    """An instance is a Cornell student"
    Instance Attributes:
    netID: student netID [str], 2-3 letters + 1-4 digits
    courses: list of courses
    major: declared major [str]
    n_credit: [int] num credits this semester
    ""
```

### Making Arguments Optional

- Can assign default values to `__init__` arguments
  - Write as assignments to parameters in definition
  - Parameters with default values are optional

**Examples**:

```python
s1 = Student("xy1234", [ ], "History")  # arguments 1,2,3
s2 = Student("xy1234", course_list)     # arguments 1 & 2
s3 = Student("xy1234", major="Art")    # arguments 1 & 3
```

```python
class Student:
    def __init__(self, netID, courses=[], major=None):
        self.netID = netID
        self.courses = courses
        self.major = major
        # < the rest of initializer goes here >
```

### Student Class Specification, v2

```python
class Student:
    """An instance is a Cornell student"
    Instance Attributes:
    netID: student netID [str], 2-3 letters + 1-4 digits
    courses: list of courses
    major: declared major [str]
    n_credit: [int] num credits this semester
    max_credit: [int] max num credits
    """
```

**New attribute!**

What do you think about this?
A look at three v2 Student instances

Anything wrong with this?

<table>
<thead>
<tr>
<th></th>
<th>id5</th>
<th>id6</th>
<th>id7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>netID</td>
<td>netID</td>
<td>netID</td>
</tr>
<tr>
<td></td>
<td>abc123</td>
<td>def456</td>
<td>gh7890</td>
</tr>
<tr>
<td></td>
<td>courses</td>
<td>courses</td>
<td>courses</td>
</tr>
<tr>
<td></td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
</tr>
<tr>
<td></td>
<td>major</td>
<td>major</td>
<td>major</td>
</tr>
<tr>
<td></td>
<td>&quot;Music&quot;</td>
<td>&quot;History&quot;</td>
<td>&quot;CS&quot;</td>
</tr>
<tr>
<td></td>
<td>n_credit</td>
<td>n_credit</td>
<td>n_credit</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>14</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>max_credit</td>
<td>max_credit</td>
<td>max_credit</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

Class Attributes

Class Attributes: Variables that belong to the Class
- One variable for the whole Class
- Shared by all object instances
- Access by <Class Name>.

Why?
- Some variables are relevant to every object instance of a class
- Does not make sense to make them object attributes
- Doesn’t make sense to make them global variables, either

Example: we want all students to have the same credit limit (Also in A4: all_of_em in both Song and M1x)

Classes Have Folders Too

Object Folders
- Separate for each instance
- Example: 2 Student objects

Class Folders
- Data common to all instances
- Not just data!
- Everything common to all instances goes here!

Functions vs Object Methods

Function: call with object as argument

- `len(my_list)`
- `print(my_list)`

Method: function tied to the object

- `my_list.count(?)`
- `my_list.sort()`
Complete Class Definition

```python
class Student:
    """Specification goes here."""
    max_credit = 20
    def __init__(self, netID, courses, major):
        ... <snip> ...
```

Python creates the Class folder after reading the class definition.

Another Method Definition

```python
c1 = Course("AEM 2400", 4)
s1.enroll(c1)
```

We now know how to make:

- Class definitions
- Class specifications
- The `__init__` function
- Attributes (using `self`)
- Class attributes
- Class methods

More Method Definitions!

```python
class Student:
    def __init__(self, netID, courses=[], major=None):
        # < init fn definition goes here>
        def enroll(self, name, n):
            # < enroll fn definition goes here>
            def drop(self, course_name):
                """removes course with name course_name from courses list
                updates n_credit accordingly
                course_name: name of course to drop [str] """
                for one_course in self.courses:
                    if one_course.name == course_name:
                        self.n_credit = self.n_credit - one_course.n_credit
                        self.courses.remove(one_course)
                        print("Just dropped " + course_name)
                        print("Currently at " + str(self.n_credit) + " credits")
```

Rules to live by (1/1)

1. Refer to Class Attributes using the Class Name

```python
s1 = Student("xy1234", [], "History")
print("max credits = " + str(Student.max_credit))
```

Name Resolution for Objects

- `myobject.myattribute` means
  - Go the folder for `myobject` `s1`
  - Find method `myattribute`
  - If missing, check `class folder`
  - If not in either, raise error

(Same thing applies to `myobject.mymethod()`)
Accessing vs. Modifying Class Variables

• **Recall:** you cannot assign to a global variable from inside a function call

• **Similarly:** you cannot assign to a class attribute from “inside” an object variable

```python
s1 = Student("xy1234", [], "History")
Student.max_credit = 23  # updates class attribute
s1.max_credit = 24       # creates new object attribute  
                          # called max_credit
```

*Better to refer to Class Variables using the Class Name*

*Just like it did in the `__init__` method!*

---

### What gets Printed? (Q)

```python
import college

s1 = college.Student("jl200", [], "Art")
print(s1.max_credit)  
s2 = college.Student("jl202", [], "History")
print(s2.max_credit)

s1.max_credit = 24
print(s1.max_credit)

print(college.Student.max_credit)
```

### What gets Printed? (A)

```python
import college

s1 = college.Student("jl200", [], "Art")
print(s1.max_credit)  
s2 = college.Student("jl202", [], "History")
print(s2.max_credit)

s2.max_credit = 23
print(s2.max_credit)

print(college.Student.max_credit)
```

### Rules to live by (2/2)

1. Refer to Class Attributes using the Class Name

```python
s1 = Student("xy1234", [], "History")
print("max credits = " + str(Student.max_credit))
```

2. Don’t forget self

- in parameter list of method (method header)
- when defining method (method body)

---

### Don’t forget self, Part 1

```python
def enroll(self, new_course):
    # if you forget self entirely
    if self.n_credit + n > Student.max_credit:
        print("Sorry your schedule is full!")
    else:
        self.courses.append(new_course)
        self.n_credit += new_course.n_credit
        print("Welcome to " + new_course.name)
```

### Don’t forget self, Part 2

```python
def enroll(self, new_course):
    # if you forget self in the body
    if self.n_credit + n > Student.max_credit:
        print("Sorry your schedule is full!")
    else:
        self.courses.append(new_course)
        self.n_credit += new_course.n_credit
        print("Welcome to " + new_course.name)
```

```python
s1 = Student("xy1234", [], "History")
c5 = Course("AEM 2400", 4)
s1.enroll(c5)
```

*TypeError: enroll() takes 1 positional arguments but 2 were given*

---

### Don’t forget self, Part 3

```python
s1 = Student("xy1234", [], "History")
c5 = Course("AEM 2400", 4)
s1.enroll(c5)
```

*NameError: global name ‘n_credit’ is not defined*