Lecture 17: Classes
(Chapters 15 & 17.1-17.5)

CS 1110
Introduction to Computing Using Python

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

• A4 due Tues Apr 13
• Prelim 2 on Apr 22 (Thurs)
• Prelim 2 seat or online session will be assigned by tomorrow via CMS. You have until Wedn Apr 14 to request a change in CMS with justification
• ACSU annual Research Night, Apr 8 5:30-7:30pm
  ▪ Interested in undergraduate research in CS?
  ▪ https://discord.com/invite/cCM3QuGY3B
Recall: Objects as Data in Folders

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

• An object is like a **manila folder**
• Contains variables
  ▪ called **attributes**
  ▪ Can change attribute values (w/ assignment statements)
• **Tab** identifies it
  ▪ Unique number assigned by Python
  ▪ Fixed for lifetime of the object
• **Type** shown in the corner
Classes are user-defined Types

Classes are how we add new types to Python

Example Classes

- Point3
- Rect
- Person
- Book
- Reader
Simple Class Definition

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 Student:

""""An instance is a Cornell student

Instance Attributes:
netID: student's netID [str], 2-3 letters + 1-4 digits
courses: nested list [ [name0, n0], [name1, n1], ... ]
    name is course name [str], n is number of credits [int]
major: declared major [str]

""""

Convention: capitalize first letter of class name
**Constructor**

- Function to create new instances
  - function name is the class name
  - Created for you automatically

- Calling the constructor:
  - Makes a new object folder
  - Initializes attributes (see next slide)
  - Returns the id of the folder

```python
courses = [['CS 1110', 4], ['MATH 1920', 3]]
s = Student('abc123', courses, 'Music')
```
What happens when constructor is called?

`s = Student("abc123", courses, "Music")`

- Creates a new object (folder) of the class Student on the heap
  - Folder is initially empty
- Executes the method `__init__`
  - if `__init__` exists
  - Puts attributes in the folder
  - Note: constructor calls `__init__` automatically if it exists
- Returns folder name, the identifier

![Diagram showing the creation of a new object in the heap](diagram.png)
def __init__(self, netID, courses, major):
    """Initializer: creates a Student
    Has netID, courses and a major
    netID: [str], 2-3 letters + 1-4 digits
    courses: nested list [[name0, n0], [name1, n1], ... ]
        name is course name [str],
        n is number of credits [int]
    major: declared major [str] """
    self.netID = netID
    self.courses = courses
    self.major = major

    courses = [ ["CS 1110", 4], ["MATH 1920", 3] ]
s = Student("abc123", courses, "Music")
# this is the call to the constructor, which calls __init__
Evaluating a Constructor Expression

\[ s = \text{Student}("abc123", \text{courses}, \text{"Music"}) \]

- Creates a new object (folder) of the class Student on the heap
  - Folder is initially empty
- Executes the method `__init__`
  - `self` = folder name = identifier
  - Other arguments passed in order
  - Executes commands in initializer
  - Note: constructor calls `__init__` automatically if it exists
- Returns folder name, the identifier
Truths about instantiating an object of a class

A) Instantiate an object by calling the constructor
B) The constructor creates the folder
C) A constructor calls the __init__ method
D) __init__ puts attributes in the folder
E) 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:**
- *Student* class: attribute *courses* must be a list

**Purpose of the class specification**
class Student:

    """Instance is a Cornell student  """
    def __init__(self, netID, courses, major):
        """Initializer: instance with netID, and courses which defaults empty
        netID: [str], 2-3 letters + 1-4 digits
        courses:   nested list [ [name0, n0], [name1, n1], ... ]
                name is course name [str], n is number of credits [int]
        major:     declared major [str]  """

        assert type(netID) == str, "netID should be type str"
        assert netID[0].isalpha(), "netID should begin with a letter"
        assert netID[-1].isdigit(), "netID should end with an int"
        assert type(courses) == list, "courses should be a list"
        assert major==None or type(major) == str, "major should be None or type str"

        self.netID = netID
        self.courses = courses
        self.major = major
Aside: The Value None

- The **major** attribute is a problem.
  - **major** is a declared major
  - Some students don't have one!

**Solution**: use value **None**
- **None**: Lack of str
- Will reassign the field later!
Making Arguments Optional

• We can assign default values to \_\_init\_ arguments
  ▪ Write as assignments to parameters in definition
  ▪ Parameters with default values are optional

Examples:

s1 = Student("xy1234", [ ], "History")  # all 3 arguments given
s1 = Student("xy1234", course_list)    # netID, courses given, major defaults to None
s1 = Student("xy1234", major="Art")    # netID, major given, courses defaults to [ ]

class Student:
    def \_\_init\_\_(self, netID, courses=[ ], major=None):
        self.netID = netID
        self.courses = courses
        self.major = major
        # < the rest of initializer goes here >
We know how to make:

- Class definitions
- Class specifications
- The \_\_init\_\_ method
- Attributes (using \texttt{self})
Continue developing our class Student …

What if we want to track **and limit** the number of credits a student is taking....

Anything wrong with this?
Class Attributes: Variables that belong to the Class

- One variable for the whole Class
- Shared by all object instances
- Access by `<Class Name>..<attribute-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
class Student:
    """Instance is a Cornell student """
    max_credit = 22
    def __init__(self, netID, courses, major):
        # < specs go here >
        # < assertions go here >
        self.netID = netID
        self.courses = courses
        self.major = major
        self.n_credit = 0
        for one_course in courses:
            self.n_credit = self.n_credit + one_course[1]  # add up all the credits

    assert self.n_credit <= Student.max_credit, "over credit limit"

Refer to class attribute using class name
Classes Have Folders Too

Object Folders

- Separate for each instance
- Example: 2 Student objects

```
Student

id5

netID 'abc123'
courses id2
major "Music"
n_credit 15
```

```
Student

id6

netID 'def456'
courses id3
major "History"
n_credit 14
```

Class Folders

- Data common to all instances

```
Student

max_credit 22
```

- Not just data!
- *Everything* common to all instances goes here!
Objects can have Methods

**Function:** call with object as argument

\[
\text{<function-name>}(\text{<arguments>})
\]

\[
\text{len(}\text{my_list})
\]

**Method:** function tied to the object

\[
\text{<object-variable>.<function-call>}
\]

\[
\text{my_list.count(7)}
\]

- **Attributes** live in **object** folder
- **Class Attributes** live in **class folder**
- **Methods** live in **class folder**
**Complete Class Definition**

- **Keyword `class`**: Beginning of a class definition.
- **Specification** (similar to one for a function):
  - `class <class-name>`:
    - """Class specification"""
    - `<assignment statements>`
    - `<method definitions>`
- **To define class variables**
- **To define class methods**
- **Example**: `class Student():
  """Specification goes here."""
  max_credit = 22
  def __init__(self, netID, courses, major):
    ... <snip> ...

- **Student max_credit 22**
  - `__init__(self, netID, courses, major)`
  - Python creates after reading the class definition.
Method Definitions

Looks like a function `def`  
- But indented inside class  
- 1st parameter always `self`

Example:
`s1.enroll("AEM 2400", 4)`  
- Go to class folder for `s1` (i.e., `Student`) that’s where `enroll` is defined  
- Now `enroll` is called with `s1` as its first argument  
- Now `enroll` knows which instance of `Student` it is working with

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

    def enroll(self, cname, n):
        if self.n_credit + n > Student.max_credit:
            print("Sorry your schedule is full!")
        else:
            self.courses.append([cname, n])
            self.n_credit = self.n_credit + n
            print("Welcome to " + cname)
```
More Method Definitions!

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[0] == course_name:
                self.n_credit = self.n_credit - one_course[1]
                self.courses.remove(one_course)
                print("just dropped "+course_name)
        print("currently have "+str(self.n_credit)+" credits")
Class Gotchas... and how to avoid them

Rules to live by:

1. Refer to Class Attributes using the Class Name

   \[
   s1 = \text{Student}("xy1234", [ ], "History")
   \]

   \[
   \text{print("max credits = ", } \text{str(Student.max\_credit))}
   \]

2. Don’t forget \texttt{self}
   
   - in parameter list of method (method header)
   - when defining method (method body)
Name Resolution for Objects

- \( \langle \text{object} \rangle . \langle \text{name} \rangle \) means
  - Go the folder for \textit{object}
  - Find attribute/method \textit{name}
  - If missing, check \textit{class folder}
  - If not in either, raise error

```python
s1 = Student("xy1234", [], "History")
# finds attribute in object folder
print(s1.netID)
# finds attribute in class folder
print(s1.max_credit) ← dangerous
```
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*
Don’t forget **self**, Part 1

```python
s1 = Student("xy1234", [], "History")
s2 = Student("ab132", [], "Math")
s1.enroll("AEM 2400", 4)
```
s1 = Student("xy1234", [], "History")
s2 = Student("ab132", [], "Math")
s1.enroll("AEM 2400", 4)

What happens?
A) Error
B) Nothing, self is not needed
C) creates new local variable n_credit
D) creates new instance variable n_credit
E) creates new Class attribute n_credit

# if you forget self

class Student:
    def __init__(self, netID, courses, major):
        self.netID = netID
        self.courses = courses
        self.major = major
        # < rest of constructor goes here >

    def enroll(self, name, n):
        if self.n_credit + n > Student.max_credit:
            print("Sorry your schedule is full!")
        else:
            self.courses.append((name, n))
            self.n_credit = self.n_credit + n
            print("Welcome to " + name)
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)
s2.max_credit = 23
print(s1.max_credit)
print(s2.max_credit)
print(college.Student.max_credit)
```

A:  
22  
22  
23  
23  
23  
22  

B:  
22  
22  
23  
23  
23  
22  

C:  
22  
22  
23  
23  
22  

D:  
22  
22  
22  
22  
23  
23  

42