Recall: Objects as Data in Folders

- An object is like a manila folder
- Contains other variables
  - Variables are called attributes
  - Can change attribute values (w/ assignment statements)
- Has a “tab” that identifies it
  - Unique number assigned by Python
  - Fixed for lifetime of the object
- Has a type listed in the corner
  - nums = [2, 3, 5]
  - nums[1] = 7

Classes are user-defined Types

Classes are how we add new types to Python

Example Classes
- Point3
- Card
- Rect
- Person

Example:

```
Point3
```

```
x 2
y 3
z 5
```

Special Method: `__init__`

```
def __init__(self, last_name, ssn, boss):
    # called by the constructor
```

```
# Initialize: creates a Worker
has last_name, ssn, and boss
Pre: last_name a string, ssn an int in range 0.999999999, and boss either
a Worker or None.

self.name = last_name
self.ssn = ssn
self.boss = boss
```

```
w = Worker('Andersen', 1234, None)
```

Evaluating a Constructor Expression

```
Worker('Andersen', 1234, None)
```

1. Creates a new object (folder) of the class Worker on the heap
   - Folder is initially empty
2. Executes the method `__init__`
   - self = folder name = identifier
   - Other arguments passed in order
   - Executes commands in initializer
3. Returns folder name, the identifier

```
w = Worker('Andersen', 1234, None)
```
Classes Have Folders Too

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

Class Folders
- Data common to all instances
- Example: student objects

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

Class Folders
- Data common to all instances
- Example: student objects

Complete Class Definition
```
class Student():
    """Specification goes here."""
    enrollment = 0
    def __init__(self, name, NetID, is_auditing):
        ...<init>...  

    def __call__(self):
        """Method definition goes here."""
        return self.name

s1 = Student("Jon Li", "jl200", True)
print(s1.name)
print(s1.enrollment)  
```

Recall: Objects can have Methods

Function:
call with object as argument

```
def len(my_list)
    """Function: call with object as argument""
    return len(my_list)
```

Method:
tied to the object

```
def __init__(self, name, NetID, is_auditing)
    """Method: function tied to the object""
    self.name = name
    self.NetID = NetID
    self.is_auditing = is_auditing
```

Name Resolution for Objects

- `(object).<name>` means
  - Go the folder for object
  - Find attribute/method `name`
- If missing, check class folder
- If not in either, raise error

```
s1 = Student("Jon Li", "jl200", True)
Student.enrollment = 2  # updates class variable
```

What gets Printed? (Q)
```
import cs1110
s1 = cs1110.Student("Jon Li", "jl200", True)
print(s1.name)  
s2 = cs1110.Student("Jill Bo", "jb200", False)
print(s2.name)
s2.enrollment = 3  
print(s2.enrollment)
print(s1.name)  
```

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

```
s1 = Student("Jon Li", "jl200", True)
Student.enrollment = 2  # updates class variable
s1.enrollment = 9  
```

Better to refer to Class Variables using the Class Name

```
s1 = cs1110.Student("Jon Li", "jl200", True)
print(s1.name)  
```

Better to refer to Class Variables using the Class Name

```
s1 = cs1110.Student("Jon Li", "jl200", True)
print(s1.name)  
s2 = cs1110.Student("Jill Bo", "jb200", False)
print(s2.name)
s2.enrollment = 3  
print(s2.enrollment)
print(s1.name)  
```