http://www.cs.cornell.edu/courses/cs1110/2018sp

Lecture 17: Classes
(Chapter 15)

CS 1110

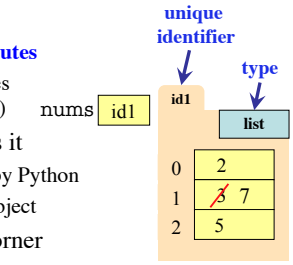
Introduction to Computing Using Python

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

Recall: Objects as Data in Folders

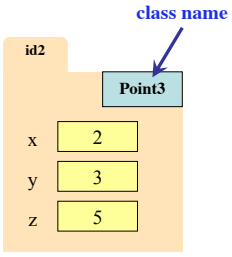
- An object is like a **manila folder**
- Contains other variables
  - Variables are called **attributes**
  - Can change attribute values (w/ assignment statements)
- Has a "tab" that identifies it
  - Unique number assigned by Python
  - Fixed for lifetime of the object
- Has a type listed in the corner

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



Classes are user-defined Types

Classes are how we add new types to Python

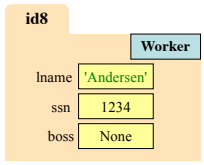


Example Classes

- Point3
- Card
- Rect
- Person

Constructors

- 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




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


Special Method: __init__

two underscores

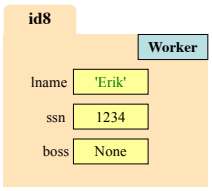
```
def __init__(self, last_name, ssn, boss): ← called by the constructor
    """Initializer: creates a Worker

    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.lname = last_name
    self.ssn  = ssn
    self.boss = boss

w = Worker('Andersen', 1234, None)   # this is the call to the constructor
                                     # the constructor calls __init__
```

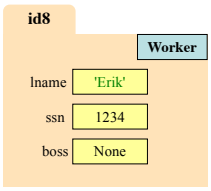
use self to assign attributes



Evaluating a Constructor Expression

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

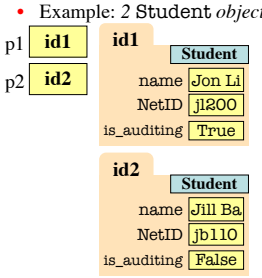
1. Creates a new object (folder) of the class Worker on the heap
   - Folder is initially empty
2. Executes the method __init__
   - self = folder name = identifier
   - Other arguments passed in order
   - Executes commands in initializer
3. Returns folder name, the identifier



Classes Have Folders Too

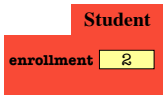
Object Folders

- 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!

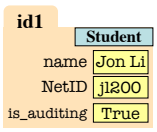
Recall: Objects can have Methods

Function: call with object as argument

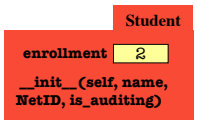
<function-name>(<arguments>)
len(my_list)

Method: function tied to the object

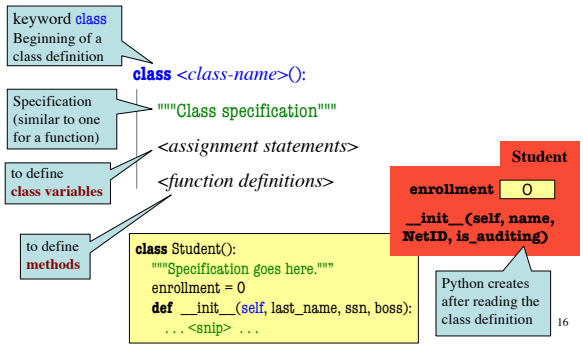
<object-variable>.<function-call>
my_list.count(7)



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

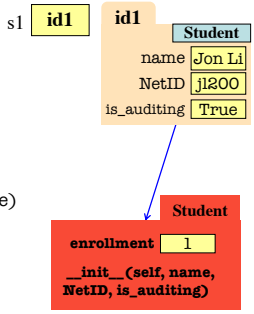


Complete Class Definition



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)
# finds attribute in object folder
print(s1.name)
# finds attribute in class folder
print(s1.enrollment) ← 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

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

Better to refer to Class Variables using the Class Name

What gets Printed? (Q)

```
import cs1110

s1 = cs1110.Student("Jon Li", "jl200", True)
print(s1.enrollment)
s2 = cs1110.Student("Jill Bo", "jb20", False)
print(s2.enrollment)
s2.enrollment = 3
print(s1.enrollment)
print(s2.enrollment)
print(cs1110.Student.enrollment)
```

A:	B:
1	1
2	2
3	3
3	3
3	2

C:	D:
1	1
2	1
2	3
3	3
2	3