Lecture 7: Objects

(Chapter 15)

CS 1110

Introduction to Computing Using Python

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

- Try out the questions on slide 28 & 35!
  - Put them in the python tutor!
  - Look at the solutions posted on the Lecture Materials
- We did not get to Slide 43 and will cover this on Thursday.

Announcements

- OKAY to show staff your code, just not other students who are not in your group
- *Per the A1 instructions:*
  - Don't submit on CMS until you form your group **on CMS**
  - If you did submit before you grouped on CMS, send email to cs1110-staff with the subject "A1 group" Make sure to cc-the person you want to be grouped with as an acknowledgement that the group formation request is reciprocated.

Be sure to start A1 now

- **Start A1 now ☺**
  - Give yourself time to think through any difficult parts
  - Consulting/office hours not too busy now—can get help fast
  - *There's time to schedule a 1-on-1 appt*

  ⇨ Rewarding learning experience
- Start A1 the night before due date
  - No time to deal with "sudden" difficulties
  - Consulting/office hours very crowded—looonnng wait time

  ⇨ Stressful experience undermines learning

Type: set of values & operations on them

Type **float:**
- Values: real numbers
- Ops: +, -, *, /,//, **,%

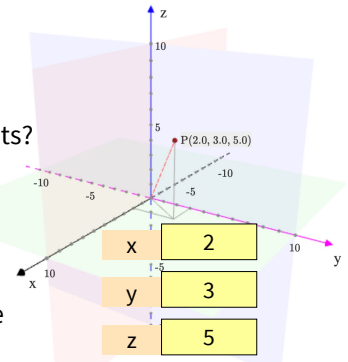
Type **int:**
- Values: integers
- Ops: +, -, *, //, %, **

Type **bool:**
- Values: `True`, `False`
- Ops: `not`, `and`, `or`

Type **str:**
- Values: strings
  - Double quotes: `"abc"`
  - Single quotes: `'abc'`
- Ops: + (concatenation)

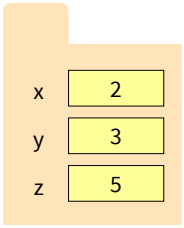
Built-in Types are not "Enough" (1)

- Want a point in 3D space
  - We need three variables
  - x, y, z coordinates
- What if we have lots of points?
  - Vars x0, y0, z0 for first point
  - Vars x1, y1, z1 for next point
  - …
  - This can get really messy
- How about a single variable that represents a point?

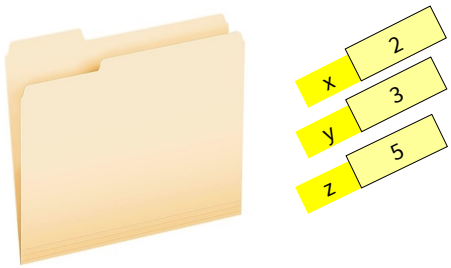


Built-in Types are not "Enough" (2)

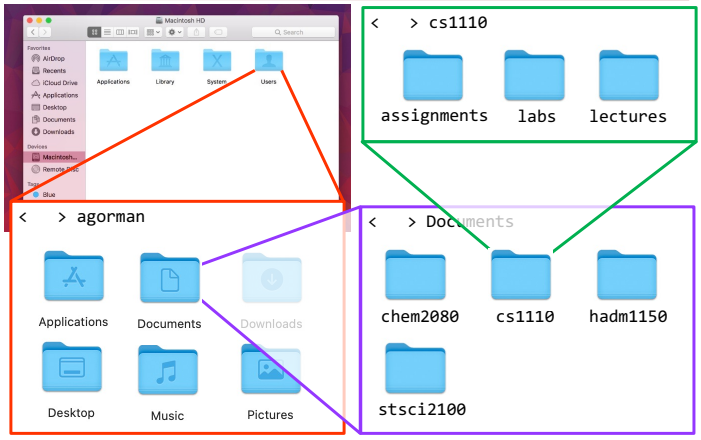
- Want a point in 3D space
  - We need three variables
  - x, y, z coordinates
- What if we have lots of points?
  - Vars x0, y0, z0 for first point
  - Vars x1, y1, z1 for next point
  - …
  - This can get really messy
- How about a single variable that represents a point?
- Can we collect them together in a "folder"?
- Motivation for objects



Analogy: A folder is used to store info (data)

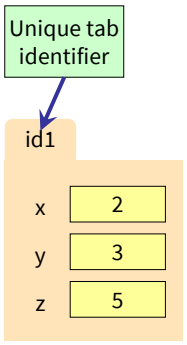


Aside: data on your computer ~~is~~ SHOULD BE!! stored in folders



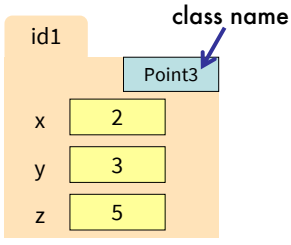
Objects: Organizing Data in Folders

- An object is like a manila folder
- It contains other variables
  - Variables are called attributes
  - These values can change
- It has an ID that identifies it
  - Unique number assigned by Python (just like a NetID for a Cornellian)
  - Cannot ever change
  - Has no meaning; only identifies



Classes: user-defined types for Objects

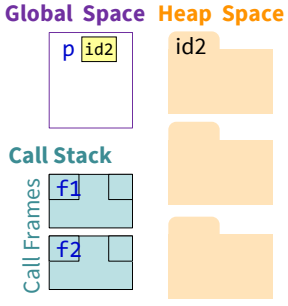
- Values must have a type
  - An object is a **value**
  - Object type is a **class**
- **Modules** provide classes
- **Example**: shapes.py
  - Defines: **Point3**, **Rectangle** classes



For now, you just need to *use* (have) the file **shapes.py**; no need to read its code yet. You can read the docstring though to learn about the **Point3** class.
Later in the course you will learn how to write such class files.

Storage in Python

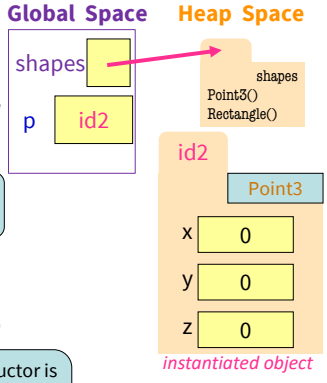
- **Global Space**
  - What you "start with"
  - Stores global variables
  - Lasts until you quit Python
- **Heap Space**
  - Where "folders" are stored
  - Have to access indirectly
- **Call Stack** (with Frames)
  - Parameters
  - Other variables local to function
  - Lasts until function returns



Constructor: Function to make Objects

Calling a Constructor Function:

- Format: ***class-name*** **(** *arguments* **)**
- **Example**: **Point3**(0,0,0)
- Makes new object (folder) w/a ***new id***
- returns folder ***id*** as value



import module with **Point3** class

Example:

```
>>> import shapes
>>> p = shapes.Point3(0,0,0)
```

SHOW IN PYTHON TUTOR !

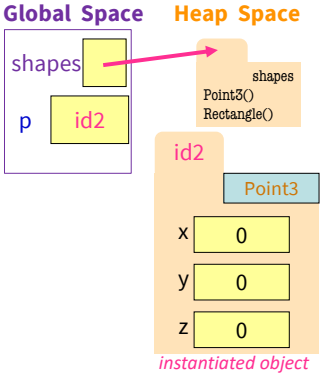
Constructor is a function. Access via module name.

Making our drawings less busy

We won't always draw module variables & module folders.

Just like we don't draw all the built-in functions.

Speaking of which…



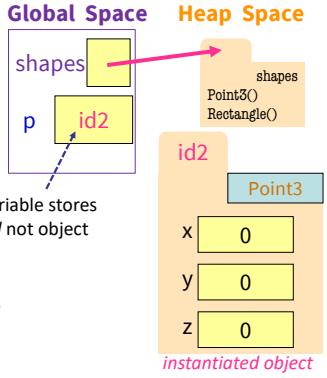
Example:

```
>>> import shapes
>>> p = shapes.Point3(0,0,0)
```


id is real!

New Built-in Function `id()`

Sometimes instead of making up an id#, we just use an arrow.



Variable stores *id* not object

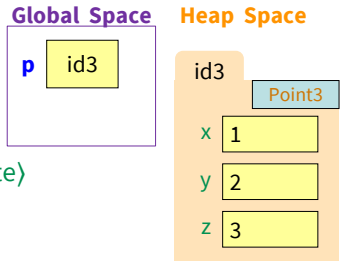
Example:

```
>>> import shapes
>>> p = shapes.Point3(0,0,0)
>>> id(p)
4371417664
>>>
```

Shows the id of p

Accessing Attributes

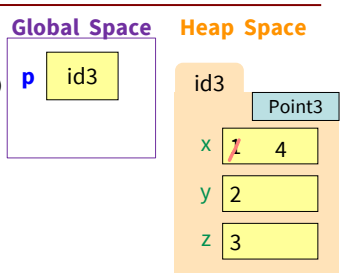
- Attributes are variables that live inside of objects
  - Can **use** in expressions
  - Can **assign** values to them
- Format: ⟨variable⟩ . ⟨attribute⟩
  - Example: `p.x`
  - Look like module variables
- To evaluate `p.x`, Python:
  1. finds folder with id stored in **p**
  2. returns the value of **x** in that folder



Accessing Attributes Example

Example:

```
p = shapes.Point3(1, 2, 3)
p.x = p.x + 3
```



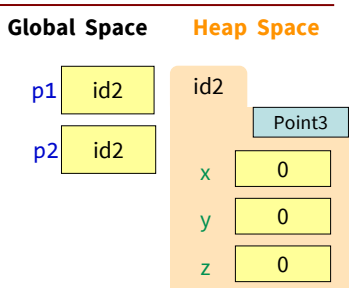
Note: we haven't drawn the module variable "shapes" or the module folder for "shapes" but they are technically there

Object Variables

- Variable stores object id
  - Reference to the object
  - Reason for folder analogy
- Assignment uses object id
  - Example:

```
p1 = shapes.Point3(0, 0, 0)
p2 = p1
```

  - Takes contents from **p1**
  - Puts contents in **p2**
  - Does not make new folder!



This is the cause of many mistakes when starting to use objects

Attribute Assignment (Question)

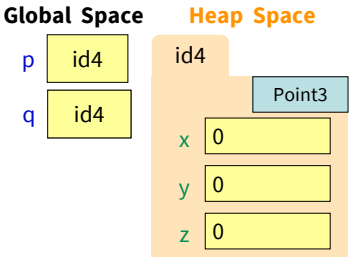
```
>>> p = shapes.Point3(0,0,0)
>>> q = p
```

- Execute the assignments:

```
>>> p.x = 5
>>> q.x = 7
```

- What is value of p.x?

A: 5
B: 7
C: id4
D: I don't know

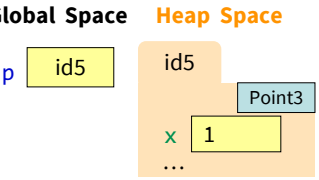


Call Frames and Objects (1)

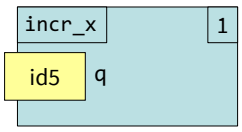
- Objects can be altered in a function call
  - Object variables hold ***id***s!
  - Folder can be accessed from global variable or parameter
- **Example**:

```
def incr_x(q):
1   q.x = q.x + 1
```

```
>>> p = shapes.Point3(1, 2, 3)
>>> incr_x(p)
```



Call Stack (w/1 Frame)

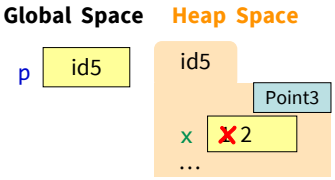


Call Frames and Objects (2)

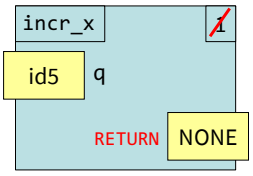
- Objects can be altered in a function call
  - Object variables hold ***id***s!
  - Folder can be accessed from global variable or parameter
- **Example**:

```
def incr_x(q):
1   q.x = q.x + 1
```

```
>>> p = shapes.Point3(1, 2, 3)
>>> incr_x(p)
```



Call Stack (w/1 Frame)



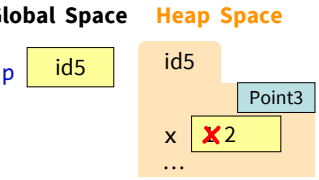
Call Frames and Objects (3)

- Objects can be altered in a function call
  - Object variables hold ***id***s!
  - Folder can be accessed from global variable or parameter

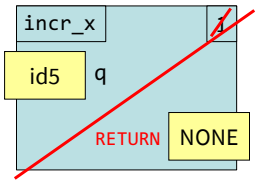
Example:

```
def incr_x(q):
1   q.x = q.x + 1
```

```
>>> p = shapes.Point3(1, 2, 3)
>>> incr_x(p)
```



Call Stack (empty)



How Many Folders (Question)

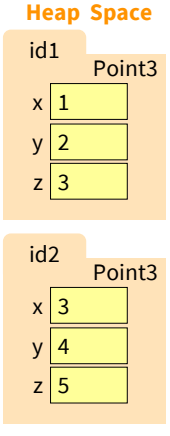
```
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
```

Draw everything that gets created (excluding the module variable & module folder).
How many folders get drawn?

What Else Gets Drawn? (Question)

```
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
```

Draw everything that gets created (excluding the module variable & module folder).
What else gets drawn?

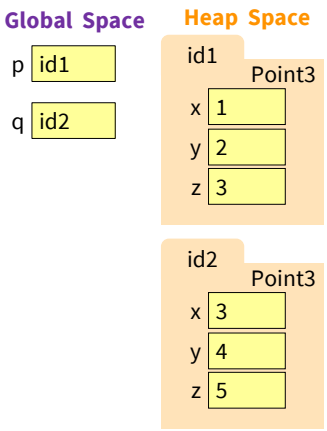


Swap Attributes (Question)

```
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

def swap_x(p, q):
1   t = p.x
2   p.x = q.x
3   q.x = t

swap_x(p, q)
```



What is in `p.x` at the end of this code?

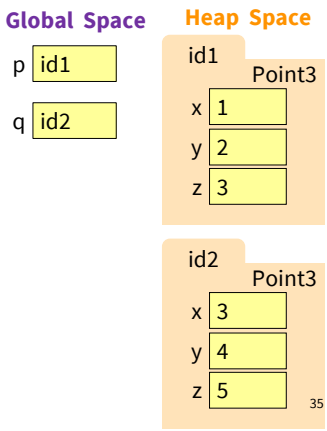
A: 0 D: 3 CORRECT
B: 1 E: I don't know
C: 2

Global p (Question)

```
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

def swap(p, q):
1   t = p
2   p = q
3   q = t

swap(p, q)
```



What is in global p after calling swap?

A: id1
B: id2 D: 2
C: 1 E: I don't know

35

Methods: a special kind of function

Methods are:

- Defined for specific classes
- Called using objects of that class

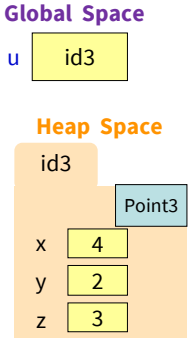
variable.method(*arguments* **)**

Example:

```
>>> import shapes
>>> u = shapes.Point3(4,2,3)
>>> u.greet()
```

"Hi! I am a 3-dimensional point located at (4,2,3)"

```
>>>
```



Where else have you seen this??

42

Recall: String Methods

- `s1.upper()`
  - Returns returns an upper case version of s_1
- `s.strip()`
  - Returns a copy of s with white-space removed at ends
- `s1.index(s2)`
  - Returns position of the first instance of s_2 in s_1
  - `error if` s_2 is not in s_1
- `s1.count(s2)`
  - Returns number of times s_2 appears inside of s_1

43

Built-in Types vs. Classes

Built-in types	Classes
• Built-into Python	• Provided by modules
• Refer to instances as *values*	• Refer to instances as *objects*
• Instantiate with simple assignment statement	• Instantiate with assignment statement with a *constructor*
• Can ignore the folders	• Must represent with folders

44

Where To From Here?

- First, <u>understand</u> **objects**
  - All Python programs use objects
  - Most small programs use objects of classes that are part of the Python Library
- Eventually, <u>create</u> your own **classes:**
  - the heart of OO Programming
  - the primary tool for organizing Python programs
- But we need to learn more basics first!

45