Py ch15: Object-Oriented Programming

31 Oct 2007
CMPT14x
Dr. Sean Ho
Trinity Western University

Quiz05: ch7-8 (10 mins, 20 pts)

- In C, why should you always allocate strings (arrays of char) to be at least one char longer than the longest string you'll need to store?
  - What could happen if you don't?
- Convert 1100 1011 from binary to both hexadecimal and octal, in Python form.
- Express 2Mb/sec in bytes/sec (binary units, not SI)
    - (you may express your answer in powers of 2)
- Write a Python function that returns a random integer between -100 and 100, inclusive.

Quiz05: answers #1-2

- In C, why should you always allocate strings (arrays of char) to be at least one char longer than the longest string you'll need to store?
  - Need to store null character to terminate string
  - If don't, won't know when to stop when reading string; may overwrite other memory when writing
- Convert 1100 1011 from binary to both hexadecimal and octal, in Python form.
  - hex: 0xCB
  - oct: 0313

Quiz05: answers #3-4

- Express 2Mb/sec in bytes/sec (binary units, not SI)
  - 2^{18} bytes/sec
- Write a Python function that returns a random integer between -100 and 100, inclusive.
  - def randint():

```
    from random import random
    return 200*int(random.random()) - 100
```

Stonybrook M2 environment

- The Stonybrook M2 software is installed on TWU lab PCs (Start->Programs->Computing)
- Stonybrook orientation: **http://twu.seanho.com/05fall/cmpt14x/stonybrook/**
- Start with an empty project file: **http://twu.seanho.com/05fall/cmpt14x/stonybrook/M2Project.sbp**
- You can have multiple programs and libraries in one project; all modules in the same project can import from one another
- Create a new program module in this project:
  - File->New Module: Program module type

Records in Python: Classes

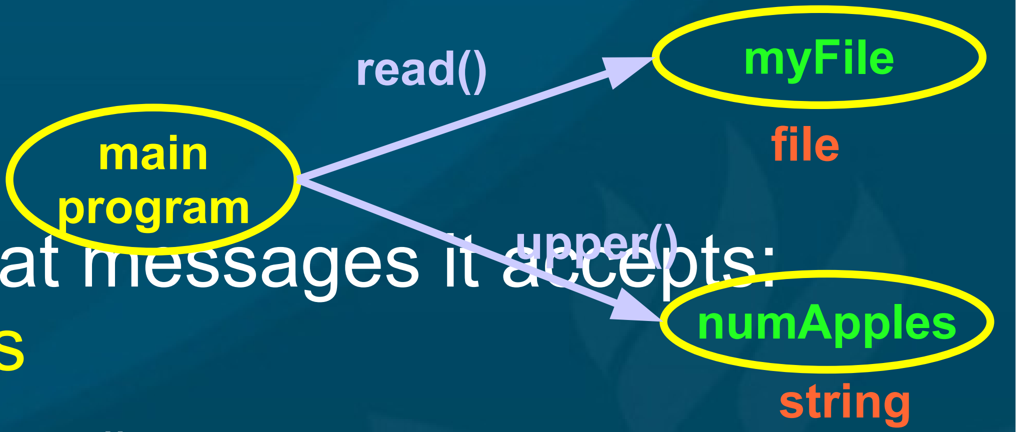
- In Python, classes are user-defined types:
    - class StudentRecord:
        - firstName = ""
        - lastName = ""
        - ID = 0
        - year = 0
  - Instantiate a new object of type StudentRecord:
    - student1 = StudentRecord()
    - student1.firstName = 'Tom'
- student1 is an instance of the class StudentRecord
  - “x is a variable of type int”

Object-oriented programming

- Procedural paradigm: programs as lists of actions
  - Focus is on the procedures (verbs)
  - Variables, data structures get passed into procedures
    - **e.g.: `string.upper('hello')`**
- Object-oriented paradigm: collections of objects
  - Focus is on the data (nouns)
  - Messages get passed between objects
  - Procedures are methods belonging to objects
    - **e.g.: `'hello'.upper()`**

Everything is an object

- In object-oriented programming, all data are objects:
  - Variables, procedures, even libraries
- We make things happen by passing messages between objects
  - myFile.read(16)
  - appleName.upper()
- The object itself defines what messages it accepts: these are called its methods
  - e.g., files have read(), write(), etc. strings have upper(), len(), etc.



Methods and attributes

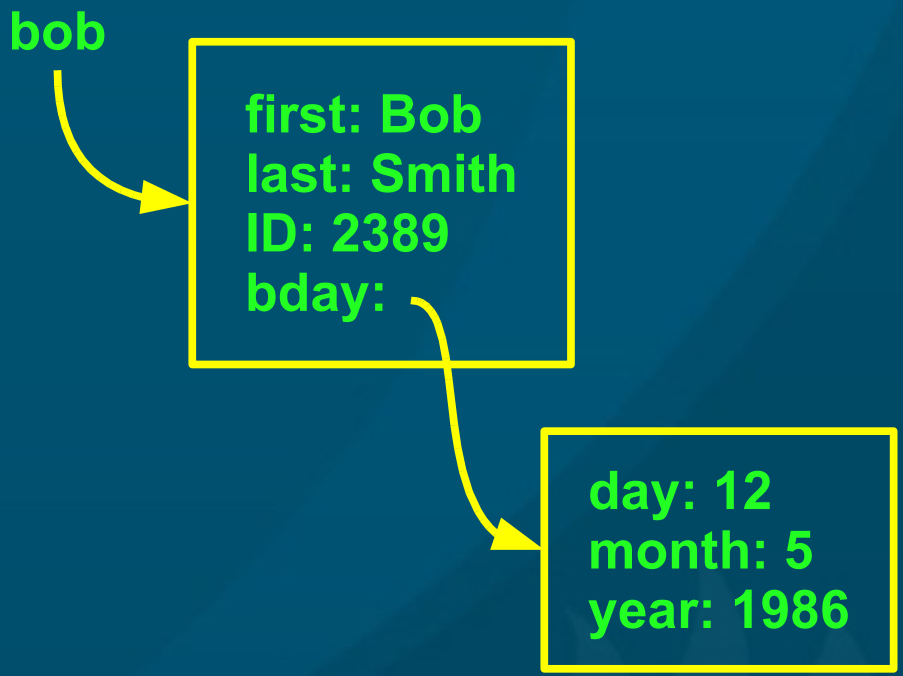
- Everything you can do with an object is encapsulated in its object definition
  - Methods make up the interface to the object
- Objects can also have attributes (variables)
- Our fractions.py ADT example:
  - Methods: get_n(), get_d(), add(), mult(), etc.
    - Everything you need to interact with a Fraction
  - Attributes: tuple (n,d)
    - Could also have two separate attributes: num, denom

Classes and instances

- We define (declare) object classes (types)
  - Attributes
  - Methods (interface)
    - Constructor and destructor
- Then we instantiate the class (declare variables)
- e.g., frac1 is a variable of type Fraction
  - frac1 is the instance,
  - Fraction is the class

More on instantiating classes

- class Date:
  - day = 0
  - month = 0
  - year = 0
- class StudentRecord:
  - firstName = ""
  - lastName = ""
  - ID = 0
  - birthdate = Date()



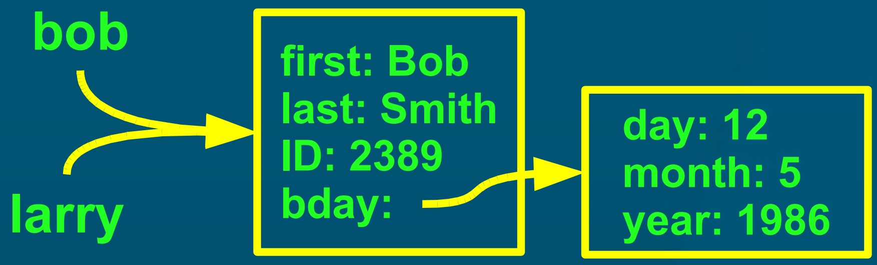
- Creating a new StudentRecord makes a new Date:
  - bob = StudentRecord()
  - bob.birthdate.year = 1986

Objects are mutable: copy vs. alias

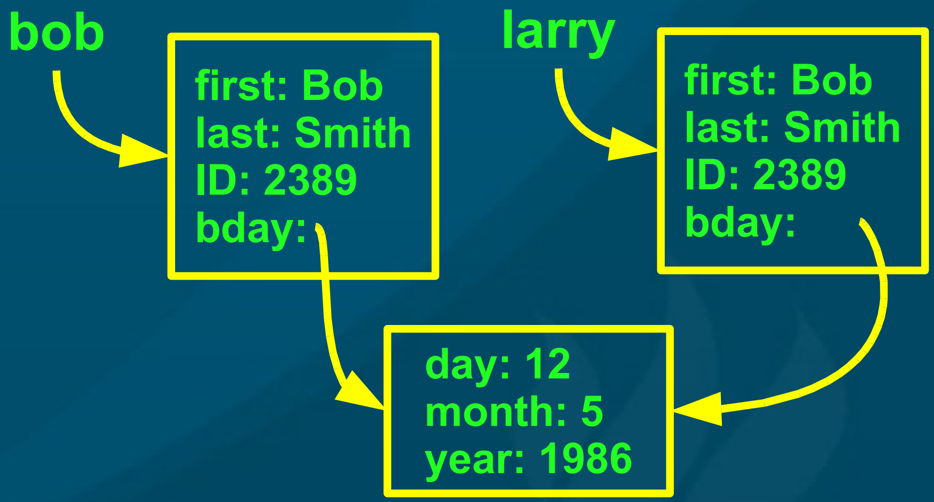
- Objects are mutable:
  - student1.ID = 25
  - student1.ID = 38
- This means assignment is just aliasing:
  - student2 = student1
  - student2.ID = 50 # affects student1.ID
- To make a separate copy, use copy.deepcopy():
  - import copy
  - student2 = copy.deepcopy(student1)
- Or create a new instance, and copy values:
  - student2 = StudentRecord()
  - student2.ID = student1.ID

More on copy vs. alias

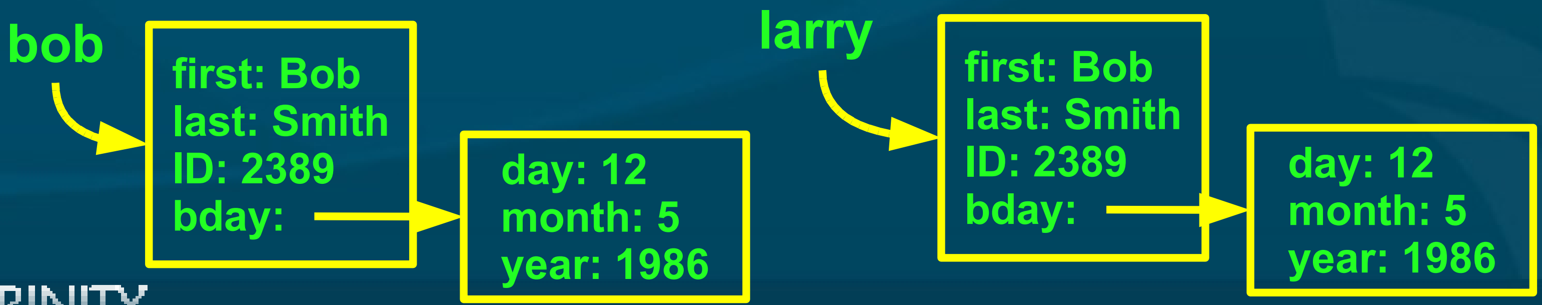
- Assignment: alias
  - larry = bob



- copy.copy(): shallow copy
  - larry = copy.copy(bob)



- copy.deepcopy(): deep copy
  - larry = copy.deepcopy(bob)



Using 'id' to look at aliases

- We can check whether two names are aliases or separate copies by using the Python built-in 'id':

```
id(student1)                          # 11563216
student2 = student1                   # alias
id(student2)                          # 11563216
student2 = copy.deepcopy(student1)  # copy
id(student2)                          # 18493888
```

Creating a list of objects

- Our student db is a list of StudentRecords
- Because of aliasing, we can't use this shortcut:
  - student = StudentRecord()
  - studentDB = [student] * 35
  - A list of 35 aliases to the same object!
- Use a for loop to create separate objects:
  - **studentDB = [0] * 35**
  - **for idx in range(len(studentDB)):**
    - **studentDB[idx] = StudentRecord()**

Review from today (Py ch15)

- Object-oriented programming paradigm
- Objects, methods, attributes
- Classes, instances
- Alias vs. shallow copy vs. deep copy

TODO items

- Register for CMPT145 if you haven't already
- Lab06 due tonight
- Lab07 due next Wed: ch9 (choose one):
  - #37+38: people db, matching
  - #40+41: online chequebook
  - #46: church directory
- Paper topic due next week Fri 9Nov
- Lab10 due last week of classes:
  - Choose one from Lab04-07, do in M2