Objects: copy vs. alias

13 Nov 2009
CMPT140
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Pretty-printing an object

- You can define the special __str__() method to return a “pretty-printed” string as a human-readable representation of your object:

```
class Student:
def __init__(.....): .... # as before
def __str__(self):
return self.first + ' ' + self.last + ', GPA: ' + self.GPA
```

- This is used by print() to display your object:

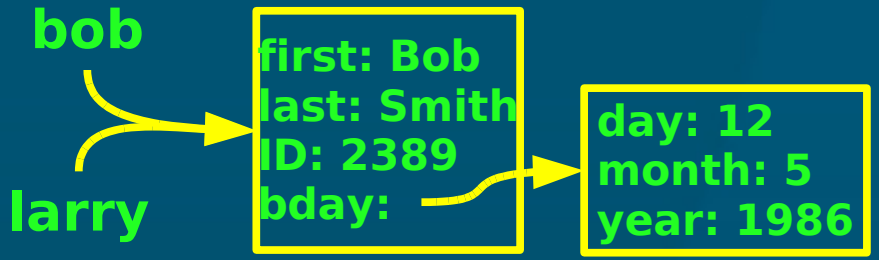
```
>>> print joe
Joe Smith, GPA: 3.8
```

Copy vs. alias for objects

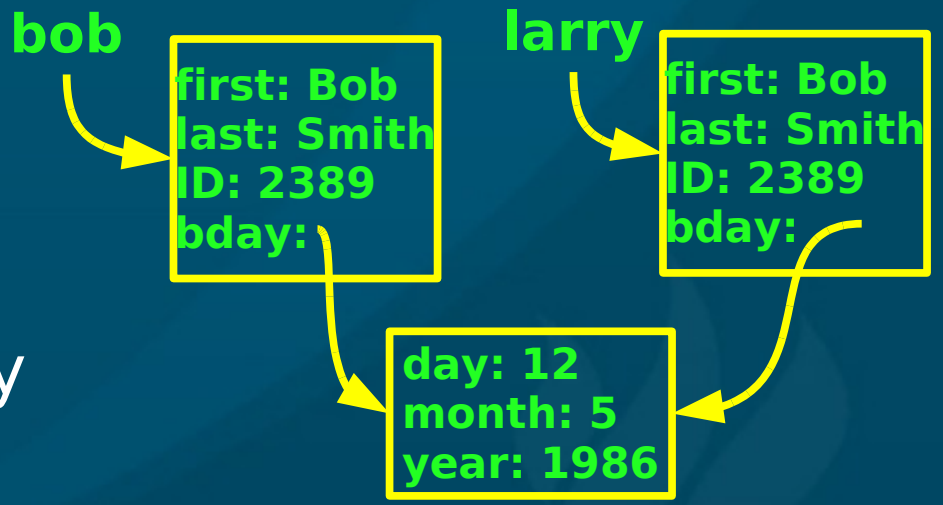
- Objects are mutable: may be modified in-place
  - **student1.GPA = 2.9**
  - **student1.GPA = 3.2**
- This means assignment is just aliasing:
  - **student2 = student1**
  - **student2.GPA = 3.4 # affects student1.GPA**
- To make a separate copy, use copy.deepcopy():
  - **import copy**
  - **student2 = copy.deepcopy(student1)**
- Or create a new instance, and copy values:
  - **student2 = Student()**
  - **student2.GPA = student1.GPA**

More on copy vs. alias

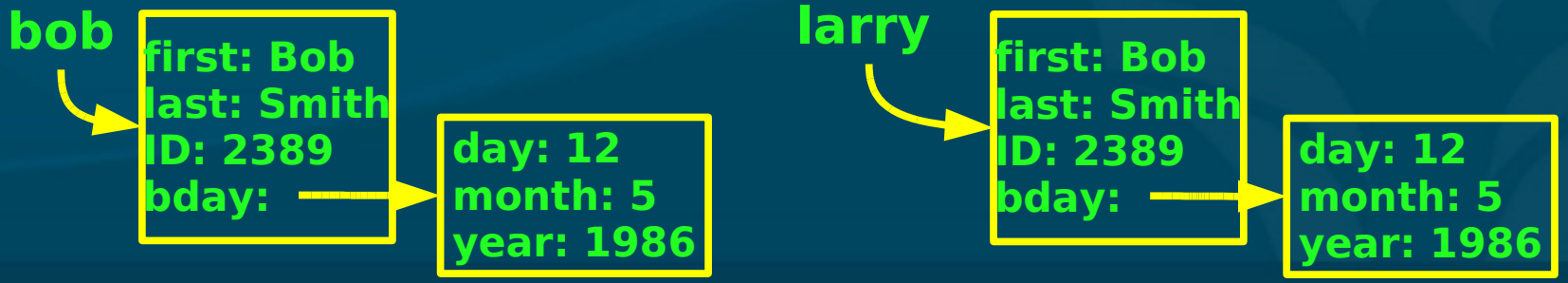
- Assignment: alias
  - **larry = bob**



- 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

- A student database is just a list of Student s
- It's tempting to use this shortcut:
  - student = Student()
  - studentDB = [student] * 35
    - But this will make 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()**