INTRO. TO OBJECT-ORIENTED PROGRAMMING IN PYTHON

Curt Clifton
Rose-Hulman Institute of Technology

Check out *PythonOOIntro* from SVN

TODAY'S PLAN

- Some notes on *scope*
- Brief introduction to syntax for objects in Python
- Remember:
  - Milestone 1 due tomorrow night
    - Don't forget Team/Language survey
  - Project Friday tomorrow, no class

PREPARATION

- In Eclipse, check out the *PythonOOIntro* project from your individual repository for the course
- Open the file *scope.py*

http://icanhascheezburger.com/category/loldog/page/3/

SCOPE IN PYTHON

- See code and comments in *scope.py* to answer quiz questions 1 and 2

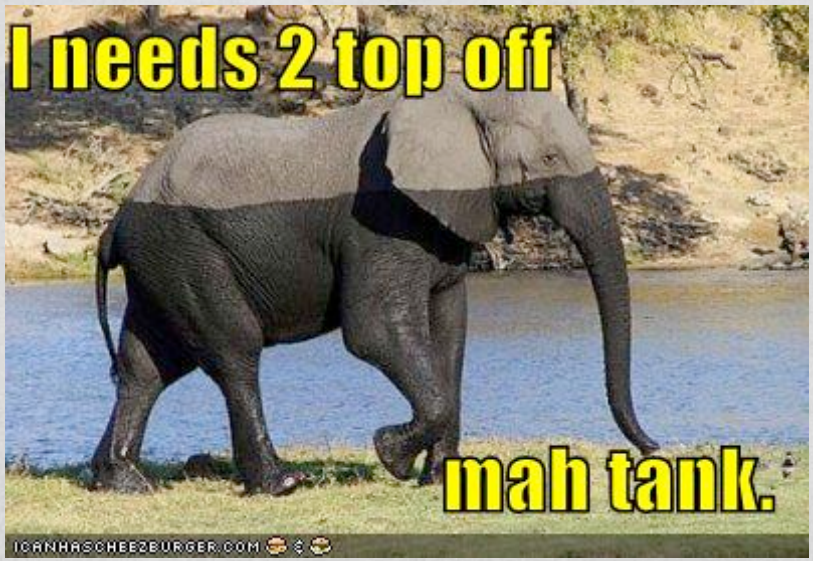
Q1,2

BUT I WANT TO ASSIGN TO THE TOP-LEVEL VARIABLE!

- You can prevent Python from creating a shadowing, local variable using *global*
- Example:

```
def fn3():
    global x
    print("x in fn3:", x)
    x = 15
    print("x in fn3:", x)
```

MUTATION != ASSIGNMENT



- Look at *fn4* and quiz question 4

Q4

IMPORT AND ALIASING

- See *scope_user.py*
- Quiz questions 5 and 6

Q5,6

BUILT-IN SCOPE

- Python doesn't keep you from assigning to built-in names
- Try this:
  - Add this code to *scope.py:*

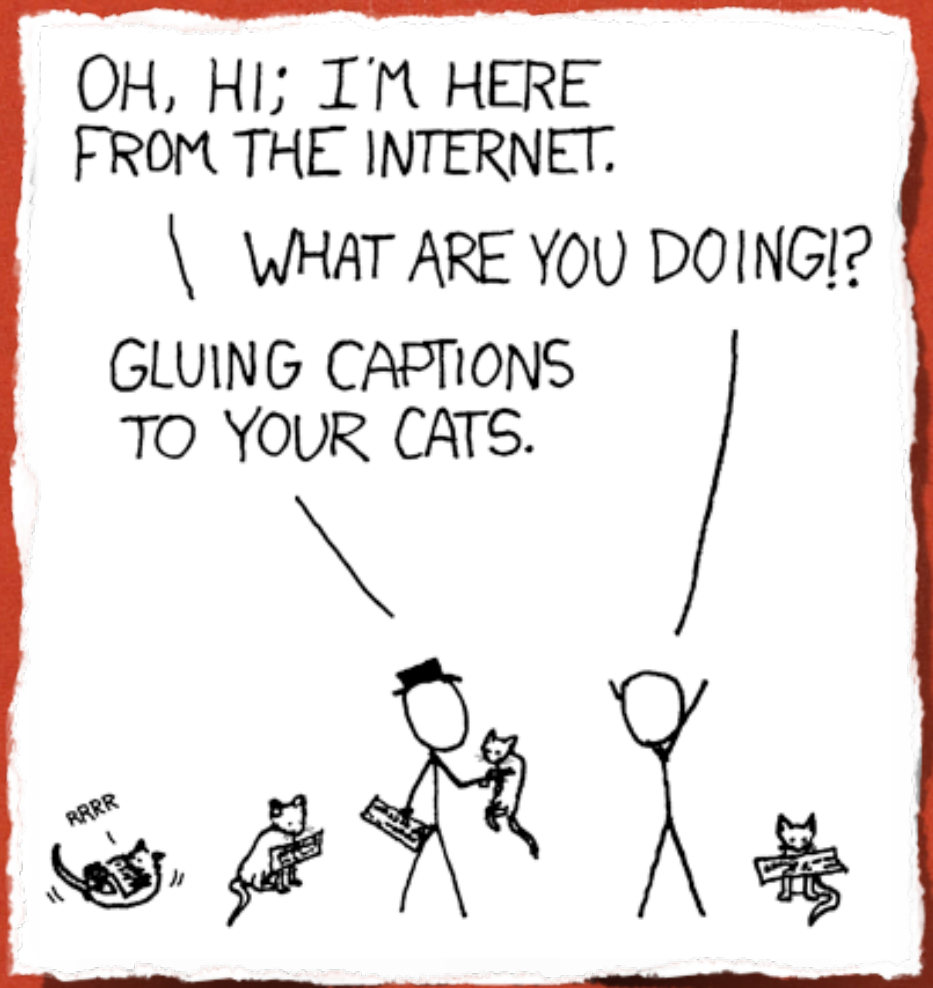
```
print(str(1))
def str(n):
    return 'boo'
print(str(1))
```

  - Run *scope.py*
  - Add `print(str(1))` to *scope_user.py* and run it
- Definition of *str* in *scope.py* **shadows** the built-in!

SCOPE SUMMARY

```
def scope_test():
    def do_local():
        spam = "local spam"
    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"
    def do_global():
        global spam
        spam = "global spam"
    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```



http://xkcd.com/262/

IN UR REALITY

HEY, AT LEAST I RAN OUT OF STAPLES.

OBJECTS IN PYTHON

- Class definitions
- Class attributes
- Instantiation
- “Fields” and “methods”

- Code for coming examples is in *class_examples.py*

CLASS DEFINITIONS

```
class ClassName:
    """Doc string."""
    # 0 or more additional statements
```

CLASS INSTANTIATION

```
class MakeMe:
    """Example for instantiation."""
    def __init__(self, x):
        self._x = x

one = MakeMe(1)
two = MakeMe(2)
print("One-two punch:", one._x, two._x)
```

FIELDS AND METHODS

- Fields
  - Like local variables, they're created by assignment
- Methods
  - Functions that "belong to" objects

```
class CountDown:
    def __init__(self, n):
        self._n = n
    def tick(self, count=1):
        self._n -= count
        if self._n <= 0:
            print('BOOM!')

counter = CountDown(5)
for i in range(8):
    counter.tick()
```

CLASSES ARE NAMESPACES

```
class Attrib:
    """Example of class attributes."""
    x, y = 2, 13

print("Attrib:", Attrib.x, Attrib.y)
```

WATCH FOR COLLISIONS

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    def x(self):
        return self.x
    def y(self):
        return self.y

p = Point(5,6)
print('p.x() = {}'.format(p.x()))
```

Error!
Why?

STATIC, CLASS, AND INSTANCE METHODS

```
class MyClass:
    """Sample class with static and class methods."""
    def __init__(self, label):
        self._label = label
    @staticmethod
    def staticFoo():
        return "static method"
    @classmethod
    def classFoo(cl):
        return "class method bound to {}".format(cl)
    def instanceFoo(self):
        return "instance method bound to {}".format(self)
    def __str__(self):
        return 'MyClass({!s})'.format(self._label)
```