CSCI 1300

Intro to Computing

Gabe Johnson

Lecture 17 Feb 22, 2013

Using Object State
Subclasses

Lecture Goals

1. Announcements
2. Test Next Friday
3. Floating Point Numbers
4. Using Object State
5. Methods, Again
6. Subclasses (not on test 2)

Upcoming Homework Assignment

HW #4 **Due: Friday, Feb 22**

Basic Objects

In this assignment we’re taking a first look at classes and objects. We’re taking slow steps, starting with making objects that we use much like dictionaries from the last assignment.

I’m doing science and I’m still alive

Ross Benbow (UW Madison) is doing science!

And he needs volunteers. ***Science needs you!***

This is a study on undergraduate STEM fields (Science, Technology, Engineering, Math). There will be pizza! But no cake.

I emailed something a while ago, and then again on Wednesday. Follow up if you can donate yourself for science (or rather, a small amount of time). *Email* ***rjbenbow@wisc.edu*** *to participate.*

No Final.

This is not news, but in case you didn’t catch it the first time, there will NOT be a final for this class. After Spring Break, we will focus on **Projects** and *(for those of you who plan on continuing to Data Structures)* **learn C++**.

Make-Up Tests

If you know you are going to miss the test:
→ **Wednesday Feb 27, 7pm in CSEL**

If you miss the test due to a legit reason:
→ **Monday March 4, 7pm in CSEL**

You must RSVP so I have enough tests. Email me as soon as you know that you can't make the actual test on Friday March 1.

Last Python Lecture

Today is the last lecture I'll be introducing concepts in Python. The test next Friday will include everything up to and including today.

On Monday we'll start learning about Java.

Floating Point Numbers

We have touched on this in passing, but I haven’t formally introduced *floating point numbers* to you yet.

An integer models a counting number. How many kids do you have? How many college degrees? How many electrons are there around an atom?

A floating point number models a continuous number. How much water is there? How tall are you? How hungry are you? What’s the average flight speed of an unladen swallow?

Floating Point Example

```
x = 15 # an integer
y = 2 # another integer
x_div_y_as_int = x / y
print x_div_y_as_int # prints 7

x = 15.0 # floating point
y = 2.0 # floating point
x_div_y_as_float = x / y
print x_div_y_as_float # prints 7.5
```

Float Literals

A floating point literal is a number that involves a decimal point. Here are examples you can recreate using the Python interactive interpreter (just type 'python' at your command prompt!)

```
>>> type(9)
<type 'int'>
>>> type(9.)
<type 'float'>
>>> type(9.0)
<type 'float'>
```

Classes vs Objects

A *class* is like a blueprint for a house. It is like a template that specifies how to build the house.

An *object* is like the house itself.

A builder can use a single blueprint to specify how to build a house. Or six. Or six hundred. There's one blueprint, but lots of houses.

Another Analogy

We have a machine that produces sprockets. We tell it how many teeth, what the inner and outer radii are, and several other things, and it spits out custom sprockets.

Use a Class to make an Object

To continue with the blueprint / house example:

```
class House:
    bedrooms = 3
    garage_stalls = 2
    bathrooms = 2
    color = “Red”

my_house = House()
your_house = House()
```

The *class* is called House. I’ll admit, this is confusing. Think of it like this: We use the class like a factory that spits out examples of itself.

Customize Objects

```
class House:
    bedrooms = 3
    garage_stalls = 2
    bathrooms = 2
    color = “Red”

my_house = House()
my_house.color = “Green”
my_house.bathrooms = 3
my_house.bedrooms = 4

your_house = House()
your_house.color = “Brown”
```

Make two houses. Make my house green with 3 baths, 4 bedrooms. Your house is brown.

Even though we don’t change the number of garage stalls in either, we still have the default number (2).

Methods

Remember that methods are just functions that:

- Are defined inside a class
- Have ‘self’ as the first parameter

```
class Dog:
    name = “Sparky”
    def bark(self):
        print self.name + “ says Woof”
```

Use that ‘self’ param

The ‘self’ parameter lets you refer to that particular object’s customized values.

```
class Dog:
    name = “Sparky”
    def bark(self):
        print self.name + “ says Woof”

d1 = Dog()
d1.name = “Minnie”
d1.bark()
d2 = Dog()
d2.bark()
```

Customize d1’s name, but not d2. This prints:

```
Minnie says Woof
Sparky says Woof
```

If you don’t use ‘self’

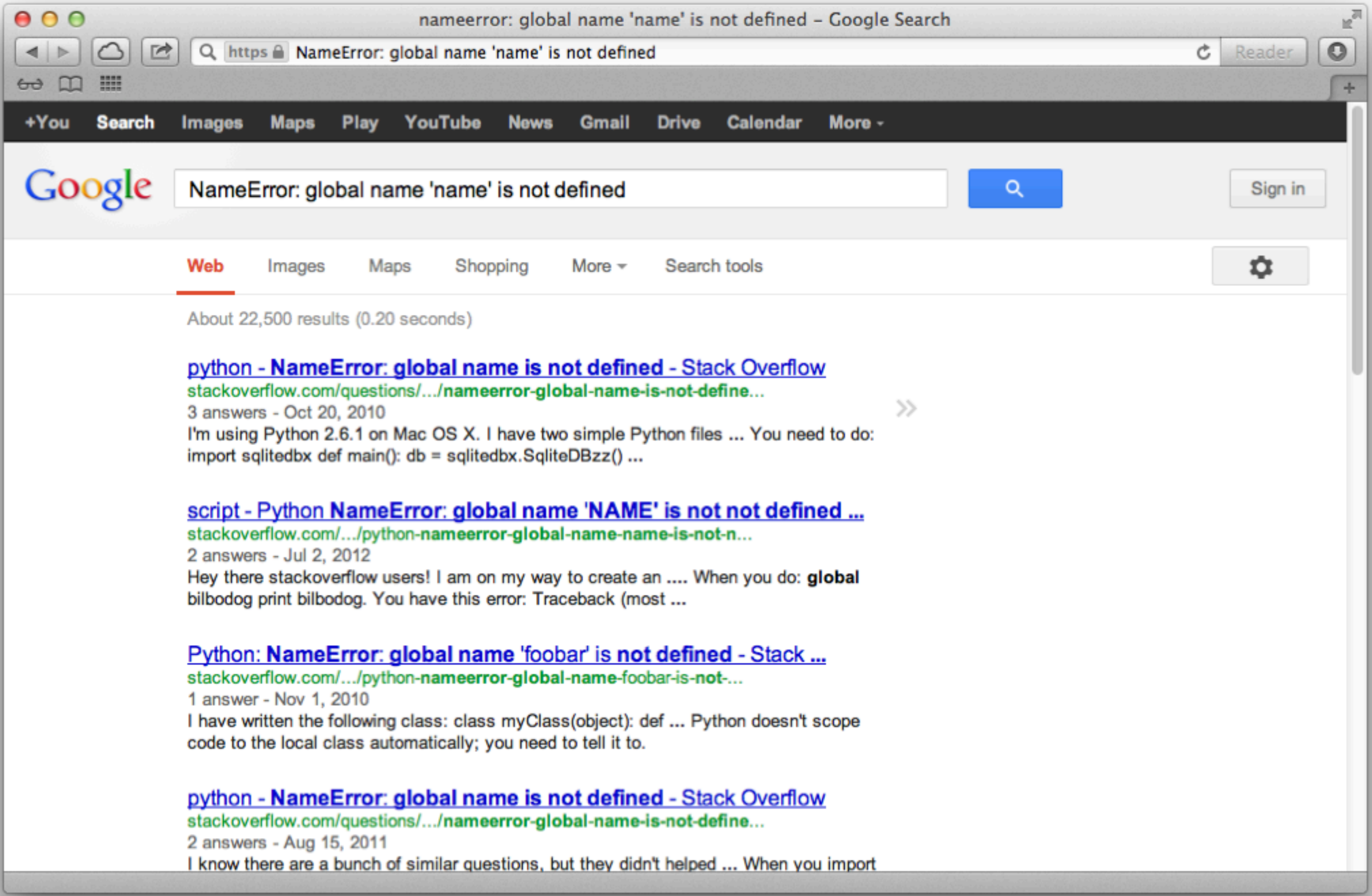
```
class Dog:
    name = “Sparky”
    def bark(self):
        print name + “ says Woof”

d1 = Dog()
d1.bark()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in bark
NameError: global name 'name' is not defined
```

What?

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in bark
NameError: global name 'name' is not defined
```

Being able to read error messages is a critical skill. When this happens and you don’t know what it means, the first and only thing you should do is copy/paste the message and paste it into Google.



nameerror: global name 'name' is not defined – Google Search

https NameError: global name 'name' is not defined

NameError: global name 'name' is not defined

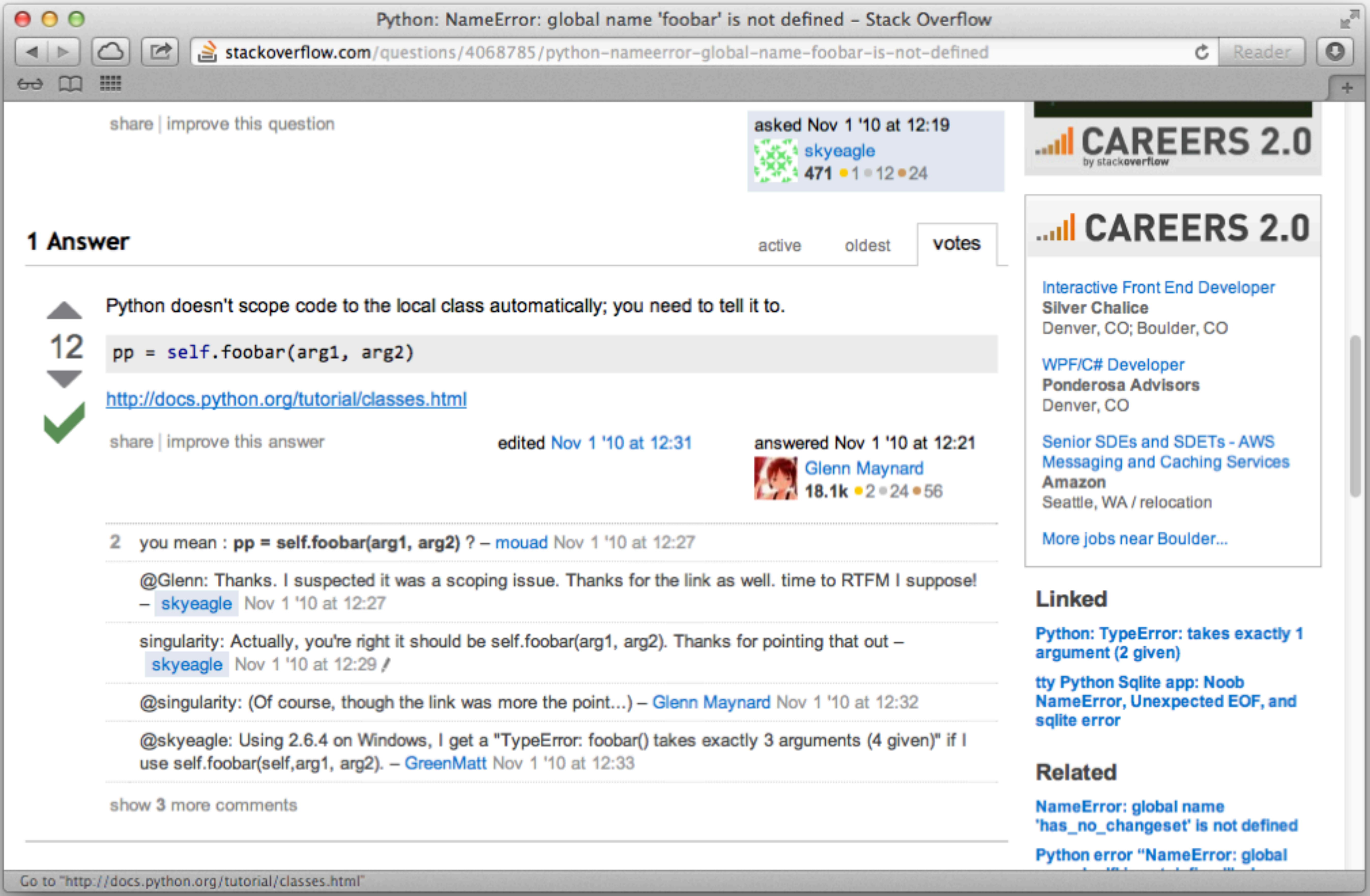
About 22,500 results (0.20 seconds)

python - **NameError: global name is not defined** - Stack Overflow
stackoverflow.com/questions/.../**nameerror-global-name-is-not-define**...
3 answers - Oct 20, 2010
I'm using Python 2.6.1 on Mac OS X. I have two simple Python files ... You need to do: import sqlitedbx def main(): db = sqlitedbx.SqliteDBzz() ...

script - Python **NameError: global name 'NAME' is not not defined ...**
stackoverflow.com/.../python-**nameerror-global-name-name-is-not-n**...
2 answers - Jul 2, 2012
Hey there stackoverflow users! I am on my way to create an .... When you do: **global** bilbodog print bilbodog. You have this error: Traceback (most ...

Python: **NameError: global name** 'foobar' is **not defined** - Stack ...
stackoverflow.com/.../python-**nameerror-global-name**-foobar-is-**not**-...
1 answer - Nov 1, 2010
I have written the following class: class myClass(object): def ... Python doesn't scope code to the local class automatically; you need to tell it to.

python - **NameError: global name is not defined** - Stack Overflow
stackoverflow.com/questions/.../**nameerror-global-name-is-not-define**...
2 answers - Aug 15, 2011
I know there are a bunch of similar questions, but they didn't helped ... When you import



Python: NameError: global name 'foobar' is not defined - Stack Overflow

stackoverflow.com/questions/4068785/python-nameerror-global-name-foobar-is-not-defined

share | improve this question

asked Nov 1 '10 at 12:19
skyeagle
471 ●1 ●12 ●24

1 Answer

active oldest votes

12

Python doesn't scope code to the local class automatically; you need to tell it to.

```
pp = self.foobar(arg1, arg2)
```

http://docs.python.org/tutorial/classes.html

share | improve this answer

edited Nov 1 '10 at 12:31

answered Nov 1 '10 at 12:21
Glenn Maynard
18.1k ●2 ●24 ●56

2 you mean : **pp = self.foobar(arg1, arg2)** ? – mouad Nov 1 '10 at 12:27

@Glenn: Thanks. I suspected it was a scoping issue. Thanks for the link as well. time to RTFM I suppose! – skyeagle Nov 1 '10 at 12:27

singularity: Actually, you're right it should be self.foobar(arg1, arg2). Thanks for pointing that out – skyeagle Nov 1 '10 at 12:29

@singularity: (Of course, though the link was more the point...) – Glenn Maynard Nov 1 '10 at 12:32

@skyeagle: Using 2.6.4 on Windows, I get a "TypeError: foobar() takes exactly 3 arguments (4 given)" if I use self.foobar(self,arg1, arg2). – GreenMatt Nov 1 '10 at 12:33

show **3** more comments

CAREERS 2.0 by stackoverflow

CAREERS 2.0

Interactive Front End Developer
Silver Chalice
Denver, CO; Boulder, CO

WPF/C# Developer
Ponderosa Advisors
Denver, CO

Senior SDEs and SDETs - AWS Messaging and Caching Services
Amazon
Seattle, WA / relocation

More jobs near Boulder...

Linked

Python: TypeError: takes exactly 1 argument (2 given)

tty Python Sqlite app: Noob NameError, Unexpected EOF, and sqlite error

Related

NameError: global name 'has_no_changeset' is not defined

Python error "NameError: global

Go to "http://docs.python.org/tutorial/classes.html"

Back in my day, we didn't have Google to answer our questions about asinine error messages. We had to slam our heads against the monitor until either we blacked out from the trauma, or we figured it out.

And we liked it!

But you should totally use Google. Pro engineers do this like ten times a day.

Subclasses (not on test 2)

The very last programming concept we are going to introduce in this class using Python is the idea of subclassing. I alluded to this last week.

We have Phones. This is a very general category.

We have Apple phones, and Samsung phones. Both are phones, but they are more specific.

Extended Subclass Example

The following sequence demonstrates a very basic (but legitimate) use of subclasses.

Lets say we want to make a text adventure game where the hero has combat with various monsters.

This whole program is 80 lines long and is up on GitHub in **monsters.py** in the code directory.

Subclasses Specialize

A subclass takes a general idea and specializes it. This might be a fairly straightforward customization. Or it might change behavior.

The hero and the monsters all have a name, attack damage, and hit points. They all also have behavior: they can attack, and take damage.

Classes Are Good For Design

Data values: *name, hit points, attack damage*. These are things we can encode with **variables**.

Behaviors: *attack,* and *take damage*. These are things we can encode with **methods**.

We will use subclasses to specialize the variables. Each subclass will have the behavior (methods) of the parent class.

Make a Base Class

```
class Monster():
    hitpoints = 10
    name = "Terrifying monster"
    attack_power = 4

    def attack(self, other):
        print self.name, "attacks", other.name
        other.take_damage(self.attack_power)

    def take_damage(self, dmg):
        self.hitpoints = self.hitpoints - dmg
        if self.hitpoints < 0:
            print "*****", self.name, "has been slain."
        else:
            print self.name, "takes", dmg, "damage"
```

Then Specialize (x 4)

```
class FluffyBunny(Monster):
    hitpoints = 5
    attack_power = 4
    name = "Cute and fluffy bunny"

class Slime(Monster):
    hitpoints = 15
    attack_power = 7
    name = "Disgusting slime thing"

class Dragon(Monster):
    hitpoints = 1000
    attack_power = 15
    name = "Ancient annoyed Dragon"

class Player(Monster):
    hitpoints = 60
    attack_power = 13
    name = "The hero"
```

Each of these classes is a *subclass* of Monster. They all *inherit* the variables and methods of the parent class. These Monster subclasses only override the variables by specializing them with custom values.

Create Instances

```
# make a list of monsters for the player
# to fight in mortal combat.
monsters = [Monster(), FluffyBunny(),
            Slime(), Dragon()]

# make a special reference to our player
player = Player()
```

Remember, ‘instance’ means ‘instance of a class’. All objects are instances of some class.

Use The Subclass Instances

```
# now play until the player is eaten by a grue.
while True:

    # loop through monsters, let them attack player
    for m in monsters:
        m.attack(player)
        print player.name, "has", player.hitpoints, "hitpoints"
        # check after each attack if the player is still
        # alive and kicking. if not, quit the game.
        if player.hitpoints <= 0:
            exit(0)

    # give our hero a chance to attack all monsters
    for m in monsters:
        player.attack(m)
        # check if the monster is dead.
        # if it is remove it from the list.
        if m.hitpoints <= 0:
            monsters.remove(m)
```

```
$ python monster.py
Terrifying monster attacks The hero
The hero takes 4 damage
The hero has 56 hitpoints
Cute and adorable fluffy bunny attacks The hero
The hero takes 4 damage
The hero has 52 hitpoints
Disgusting slime thing attacks The hero
The hero takes 7 damage
The hero has 45 hitpoints
Ancient and annoyed Dragon attacks The hero
The hero takes 15 damage
The hero has 30 hitpoints
The hero attacks Terrifying monster
***** Terrifying monster has been slain.
The hero attacks Disgusting slime thing
Disgusting slime thing takes 13 damage
The hero attacks Ancient and annoyed Dragon
Ancient and annoyed Dragon takes 13 damage
Cute and adorable fluffy bunny attacks The hero
The hero takes 4 damage
The hero has 26 hitpoints
Disgusting slime thing attacks The hero
The hero takes 7 damage
The hero has 19 hitpoints
Ancient and annoyed Dragon attacks The hero
The hero takes 15 damage
The hero has 4 hitpoints
The hero attacks Cute and adorable fluffy bunny
***** Cute and adorable fluffy bunny has been slain.
The hero attacks Ancient and annoyed Dragon
Ancient and annoyed Dragon takes 13 damage
Disgusting slime thing attacks The hero
***** The hero has been slain.
The hero has -3 hitpoints
```