

# CS 61A Discussion 6

---

March 03, 2016

# Agenda

- Quiz 6
- OOP (Object-Oriented Programming)
- OOP Inheritance
- The `nonlocal` keyword

# Quiz 6

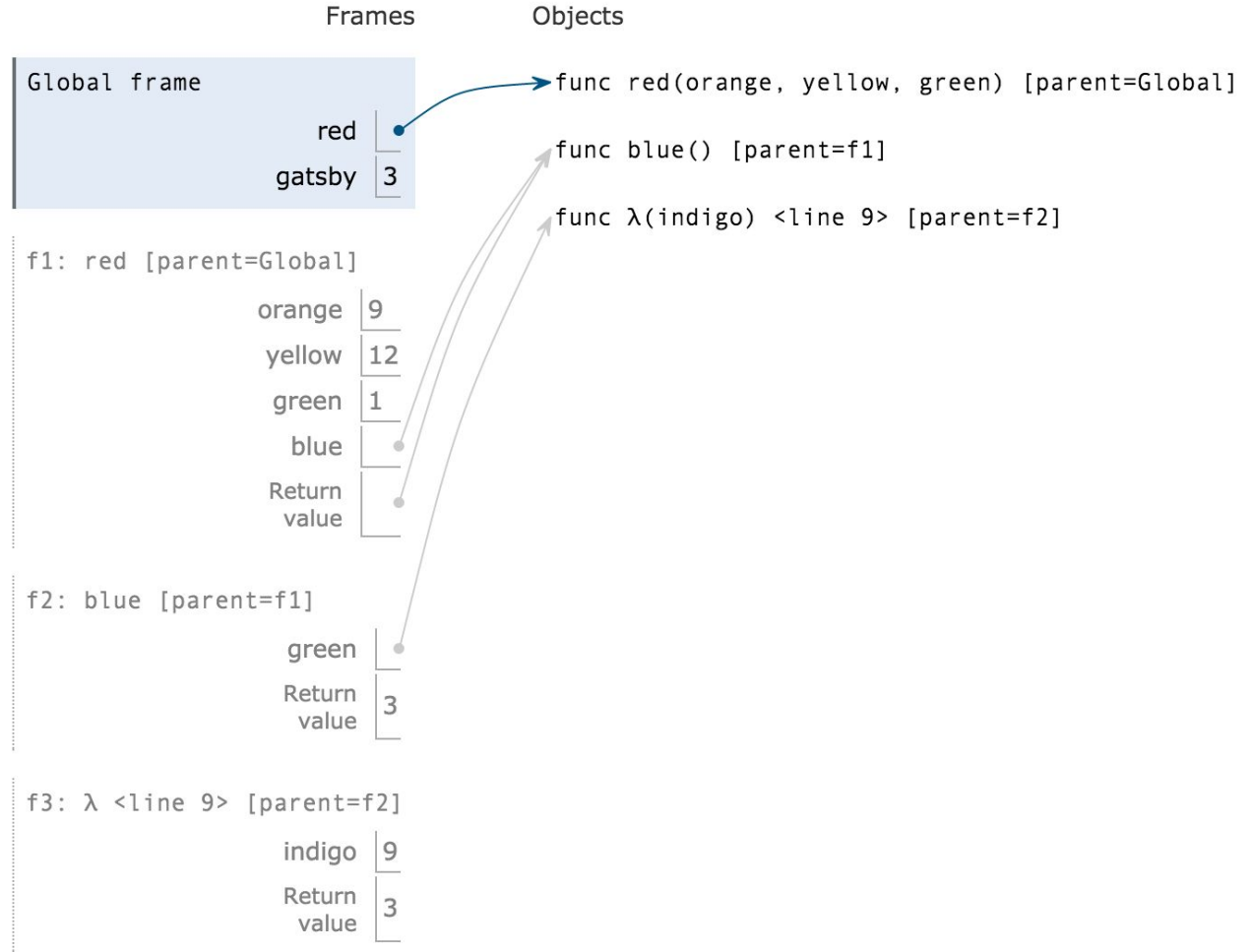
You should be able to do the quiz in under 9 minutes (this is how much time we'd allocate for you on a test).

## Things of Note

+ `x ** 0.5` returns the square root of `x` (as a float)

+ `nonlocal` works pretty much no matter where it is in the function

+ `int(x)` returns the integer form of `x`



# Object-Oriented Programming

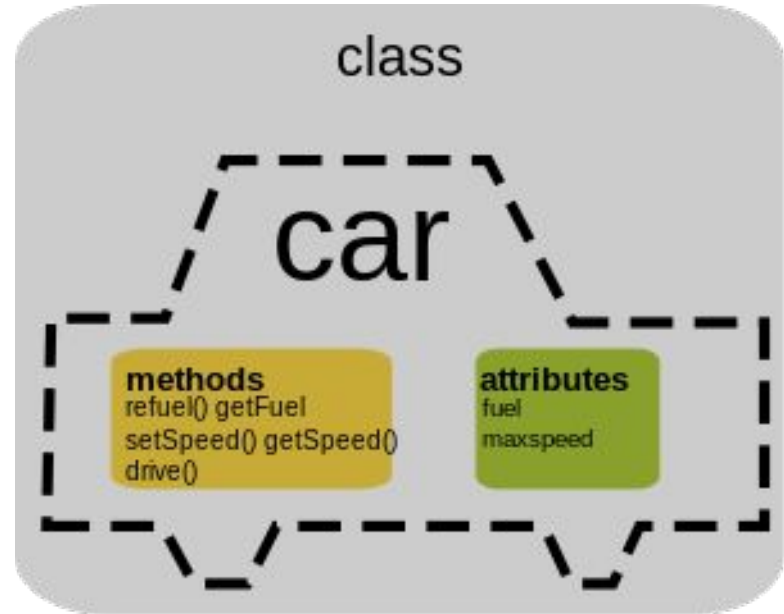
---

# OBJECTS

Objects are basically a formalization of data abstraction (an object is a programmatic representation of some real-life “thing”).

An object has **state** (attributes  $\approx$  variables) and **behavior** (methods  $\approx$  functions).

These things can be tied to an object itself, or to its *type* (i.e. its **class**).



# Things to Know

+ A **class** is like the type of an object. Formally, it's a blueprint that defines attributes and methods for an object.

+ An **object** is an instance of a class.

- Objects are created using `<classname>(...)`, which calls `<classname>`'s `__init__` function.

+ A **constructor** is the `__init__` function that creates an instance of a class (i.e. it creates objects). Normally, it just initializes variables and stuff.

## Things to Know, *cont.*

- + An **instance attribute** is specific to an instance.
- + A **class attribute** is specific to a class.



## Things to Know, *cont.*

+ A **local variable** is a variable that only exists within a function. In OOP, you can't access these variables outside of functions.

+ On the other hand, you *can* access **instance attributes** and **class attributes** outside of functions. This is done via **dot notation**, where you have the following syntax:

<OBJECT or CLASS>.<ATTRIBUTE>

If <OBJECT or CLASS> is an object and there's both an instance <ATTRIBUTE> and a class <ATTRIBUTE>, dot notation will prioritize the instance version.

# Things to Know

`self` is what you should associate with **objects** (aka instances).

- + If you have a method with arguments (`self, ...`), it's an instance method.
- + If you see `self.attribute` (this can only happen in an instance method), that's an instance attribute.
- + Note: If you see `<object>.attribute`, that's also an instance attribute.

Keep in mind that `self` is always bound to a specific object instance. This happens automatically when an instance method is called (as in `obj.method()`).

If you have an **object** on the left of the dot, then `self` will be bound to that object. If you have a **class** on the left of the dot, you'll have to explicitly pass in a `self` object.

# Mistakes and Misconceptions

- `class Foo(Bar):` ← This is not a function definition. `Bar` is not an argument here.
- When in a method, don't say `var` instead of `self.var` if you want an instance attribute.

# Inheritance

+ Inheritance is another way in which OOP models the real world.

A **subclass** is a more specific version of a **parent class**.

- ex. a Square IS A Rectangle, which IS A Shape

- ex. a Car IS A Vehicle. So is a Bus.

Subclasses inherit all of the methods and class attributes from the parent class.

They can also override attributes, or add more attributes.

The OG class is **object**, which is a parent class of everything.

`nonlocal`

---

# Nonlocality

When you say `nonlocal x`:

You're saying that in this function, `x` refers to a variable that was defined in some parent frame. When you make assignments to `x`, it will change the `x` in the parent frame.

Notes:

- `x` must be in a parent frame that ISN'T the global frame.
- If you have a `nonlocal x`, you can't have a local `x`. Any time you refer to `x` within the function, you're talking about the `x` in the parent frame.