CS61A Discussion 6: **Inheritance & Nonlocal**

TA: **Jerry Chen**
Email: **jerry.c@berkeley.edu**
TA Website: **jerryjrchen.com/cs61a**

Attendance

Form: **tinyurl.com/jerrydisc**

For the weekly question, please complete the quiz.

Agenda

1. Week in Review
2. Feedback
3. Nonlocal
4. OOP

Week In Review

Maps!

Lab6! (Nonlocal and OOP)

Hw4!

Hog composition

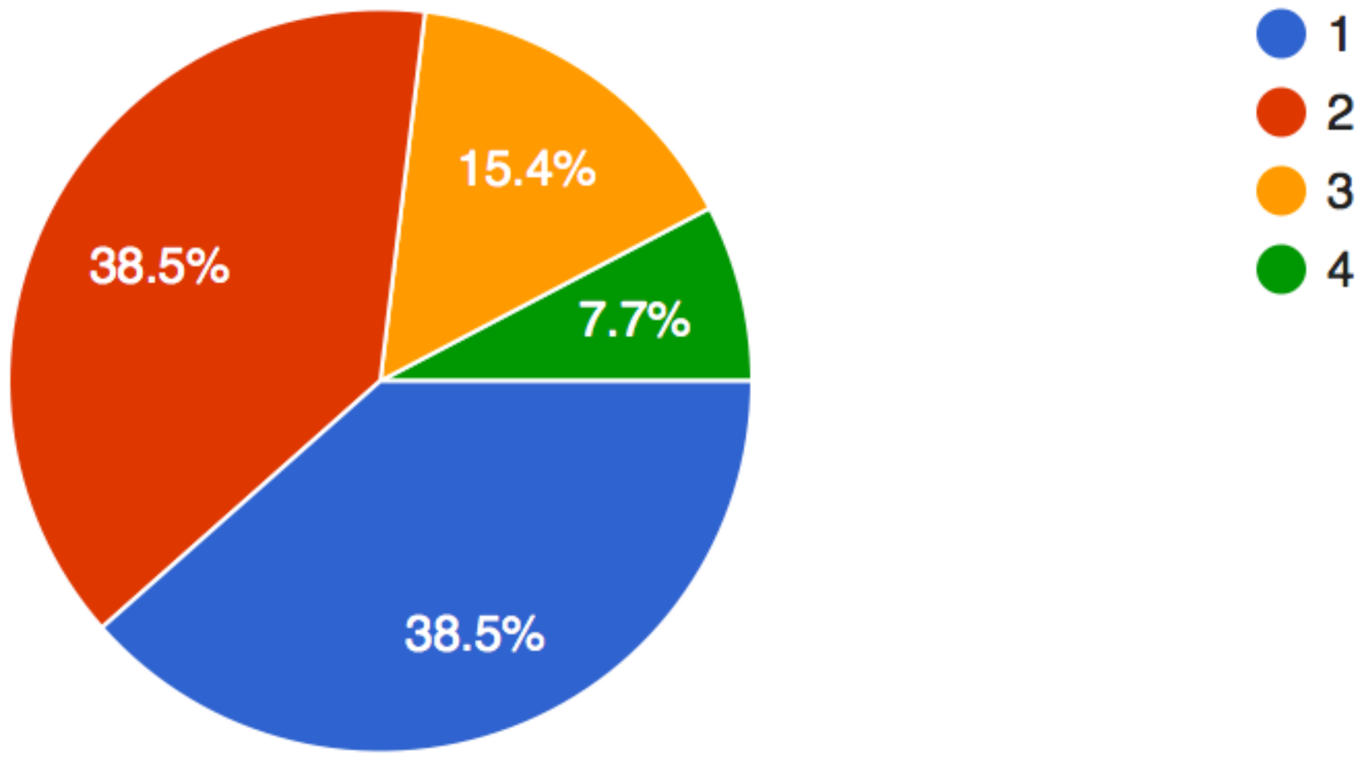
Feedback

Based on feedback:

- Some more time on problems
- Walking through a few of the shorter problems together first
- Lecture will also be more compact as a result

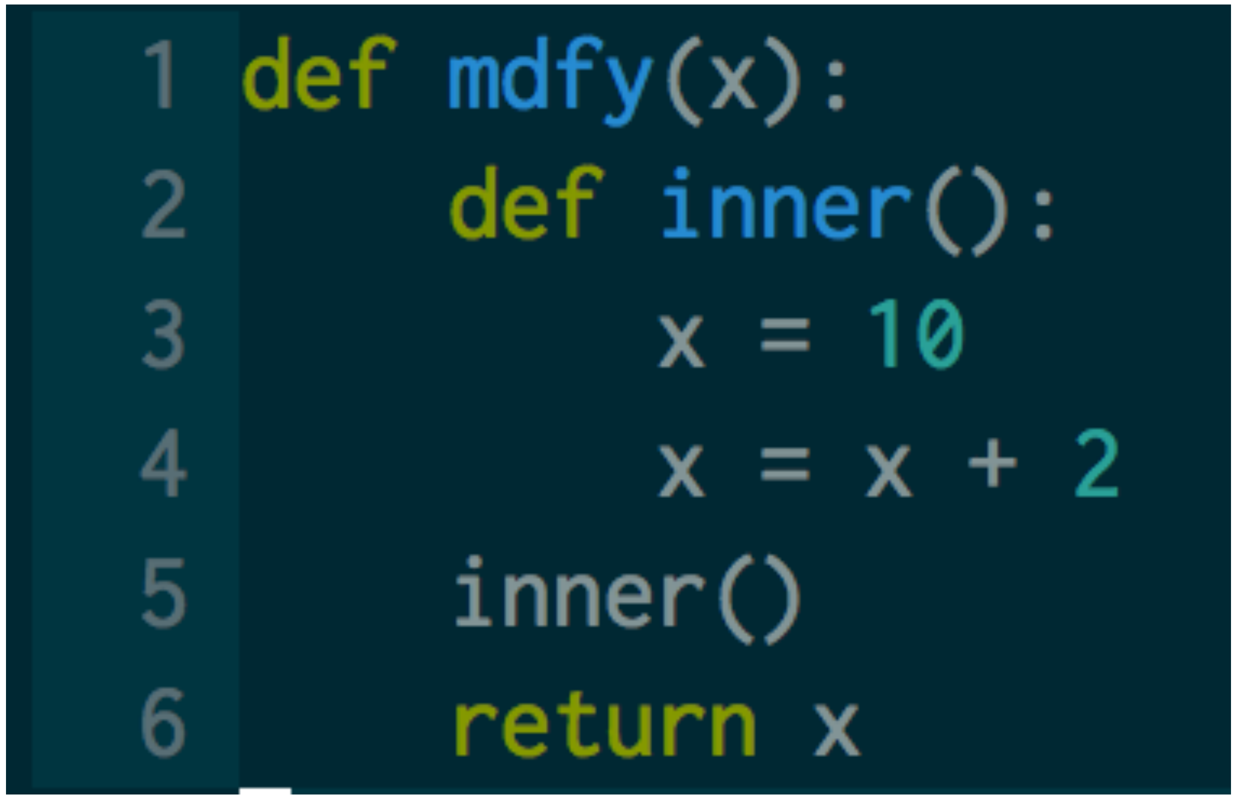
Feedback

(Optional) Which number appears at the top of this list? (13 responses)



Nonlocal

Why do we need nonlocal?
What will be the result of the output below?



```
def mdfy(x):
    def inner():
        x = 10
        x = x + 2
    inner()
    return x
```

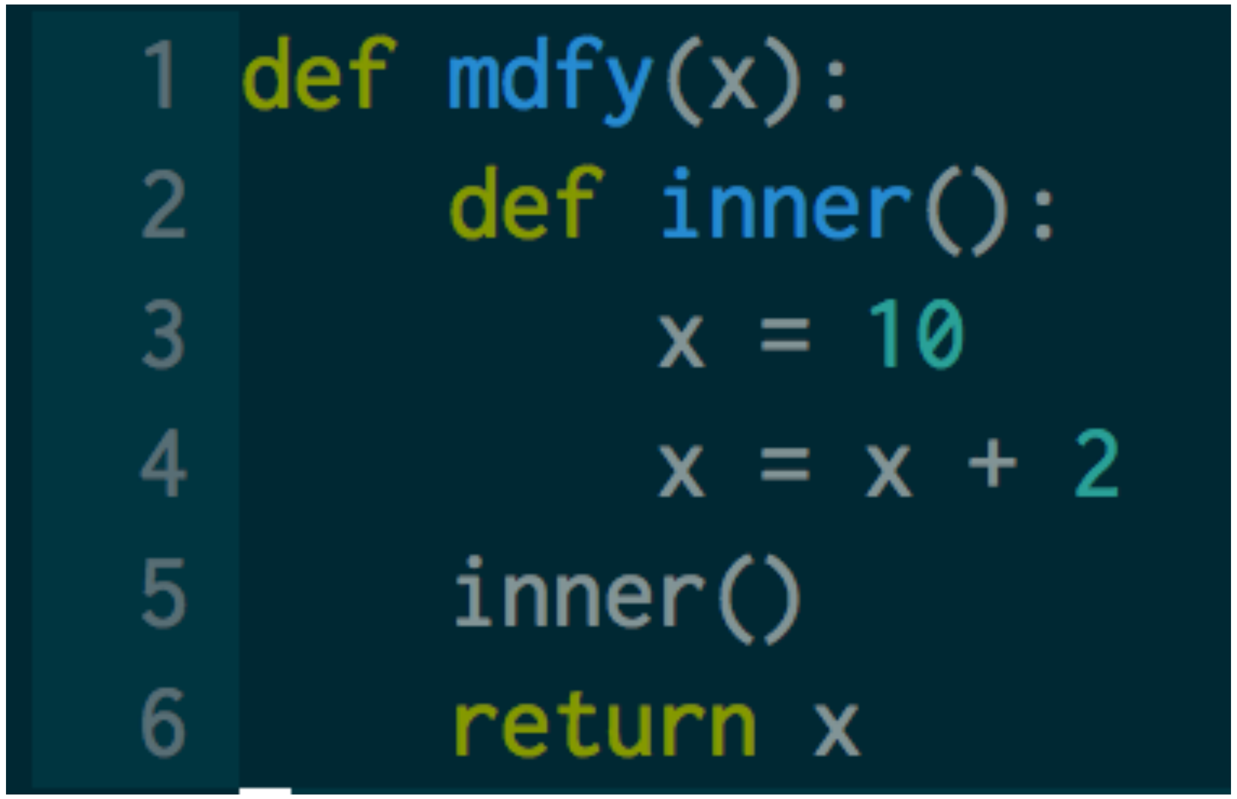
A	10
B	20
C	12
D	22
E	Error

```
>>> x = mdfy(20)
>>> x
```

____

Nonlocal

Why do we need nonlocal?
What will be the result of the output below?



```python
def mdfy(x):
    def inner():
        x = 10
        x = x + 2
    inner()
    return x
```

A	10
[✓ B]	20
C	12
D	22
E	Error

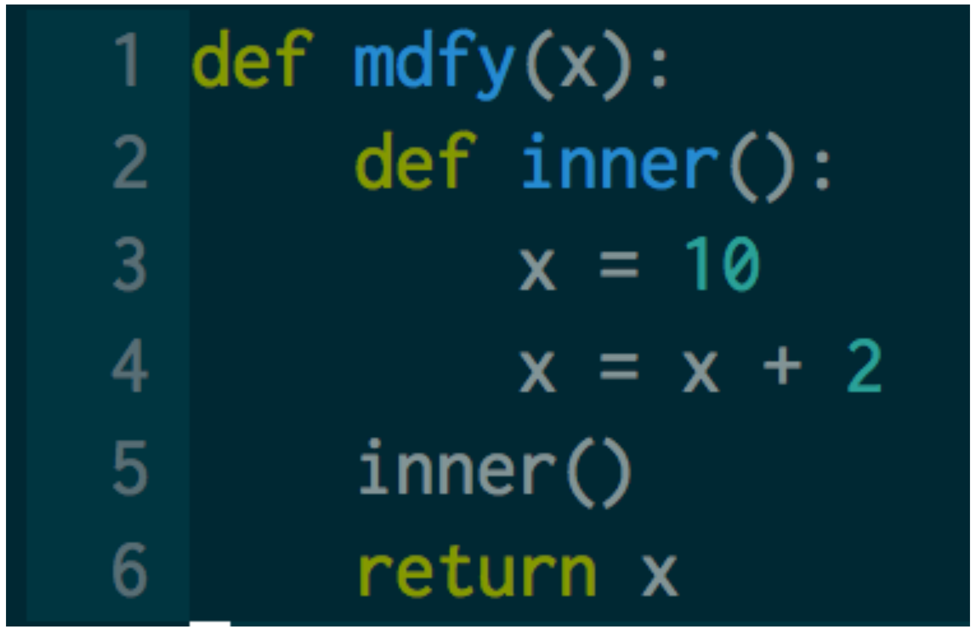
```
>>> x = mdfy(20)
>>> x
```

____

Nonlocal

What’s happening in inner()?

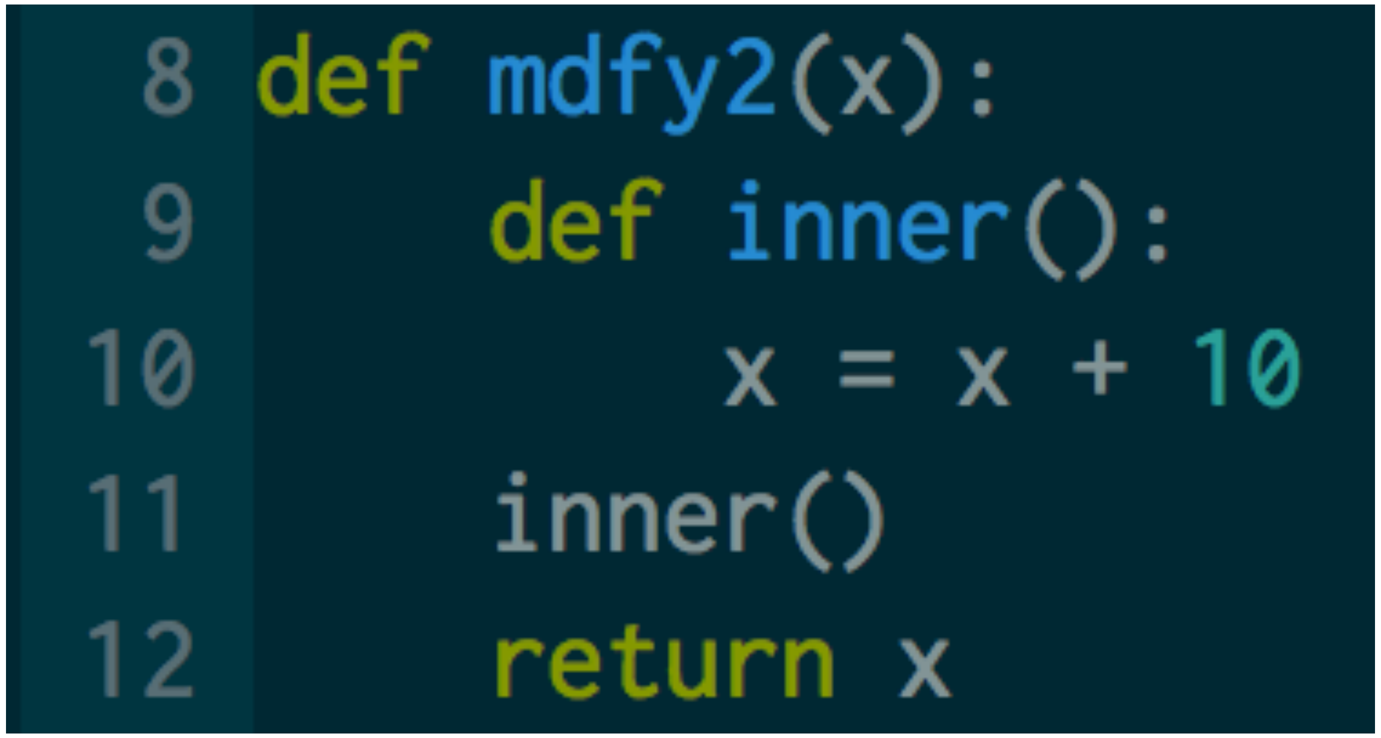
- We created a local variable x and assigned 10.
- Then, we incremented that local variable by 2.
- The one in “mdfy” is unchanged!



```python
def mdfy(x):
    def inner():
        x = 10
        x = x + 2
    inner()
    return x
```

Nonlocal

Let's try again.
What will happen here?



```python
def mdfy2(x):
    def inner():
        x = x + 10
    inner()
    return x
```

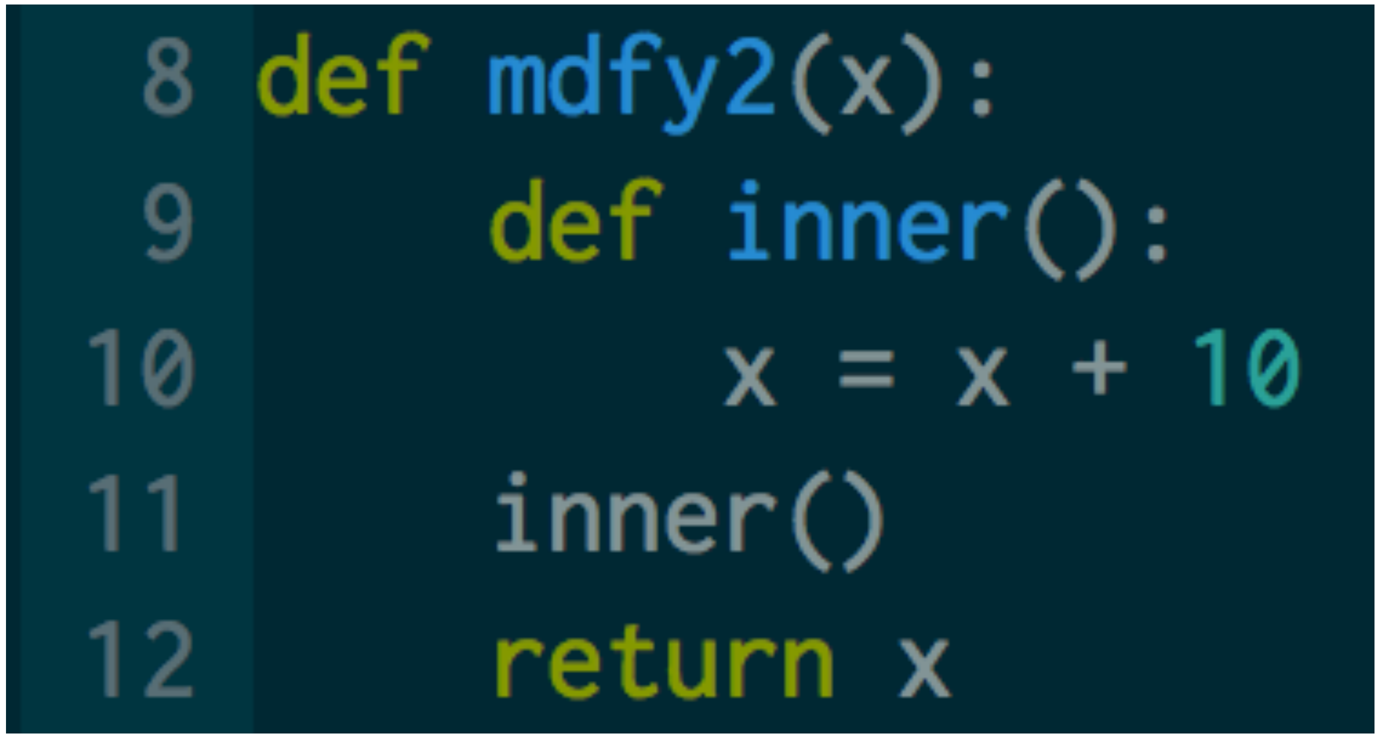
A	10
B	20
C	30
D	40
E	Error

```
>>> x = mdfy2(20)
>>> x
```

____

Nonlocal

Let's try again.
What will happen here?

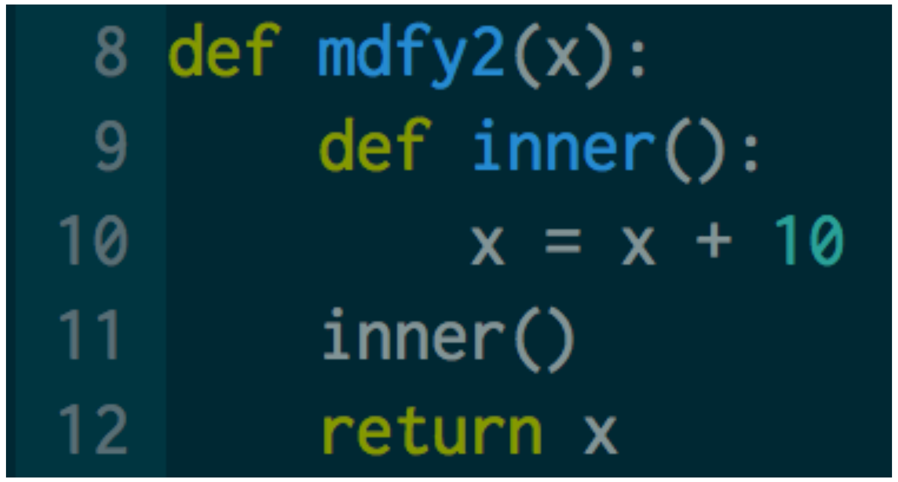


```python
def mdfy2(x):
    def inner():
        x = x + 10
    inner()
    return x
```

A	10
B	20
C	30
D	40
[✓ E]	Error

```
>>> x = mdfy2(20)
>>> x
```

Nonlocal



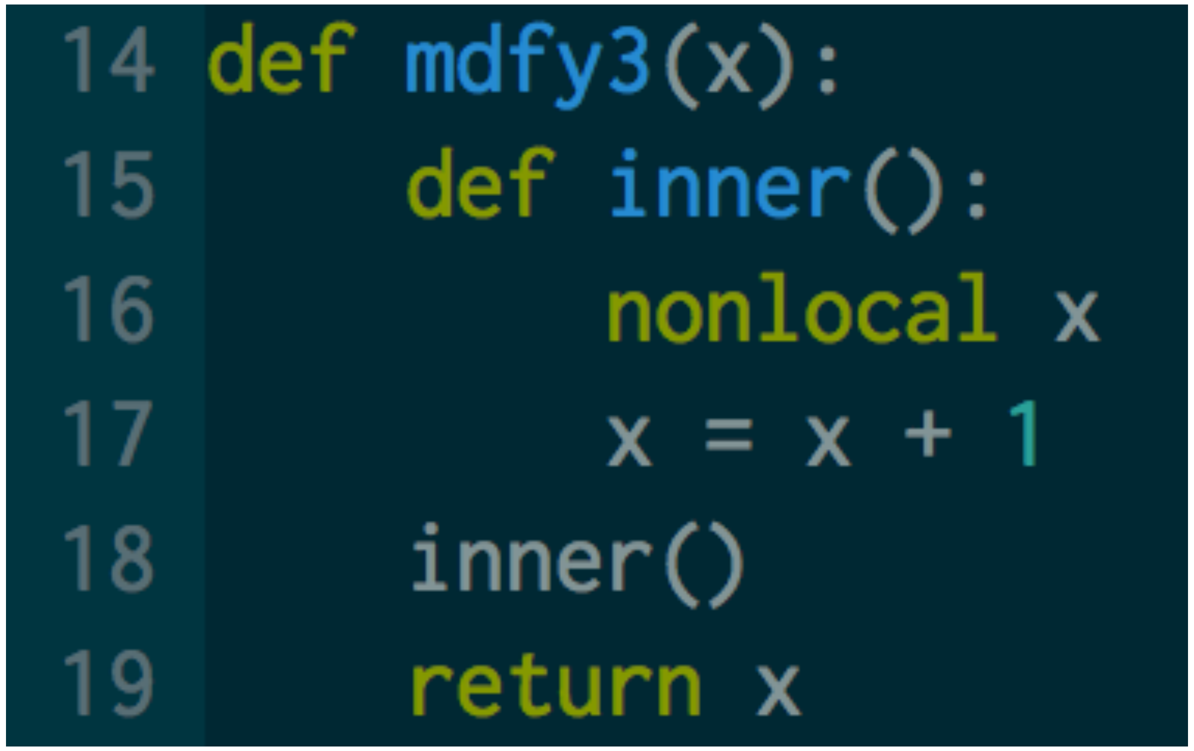
```
def mdfy2(x):
    def inner():
        x = x + 10
    inner()
    return x
```

Uh oh. This is even worse!

- **Can** lookup x from parent frame
- **Cannot** also bind to an x in the current frame
- Confusingly, this will give an “unbound local error” claiming we referenced x before assignment (Read 2.4.4 in your textbook)

Nonlocal

As you may have guessed, nonlocal is required.
Here's the proper syntax:



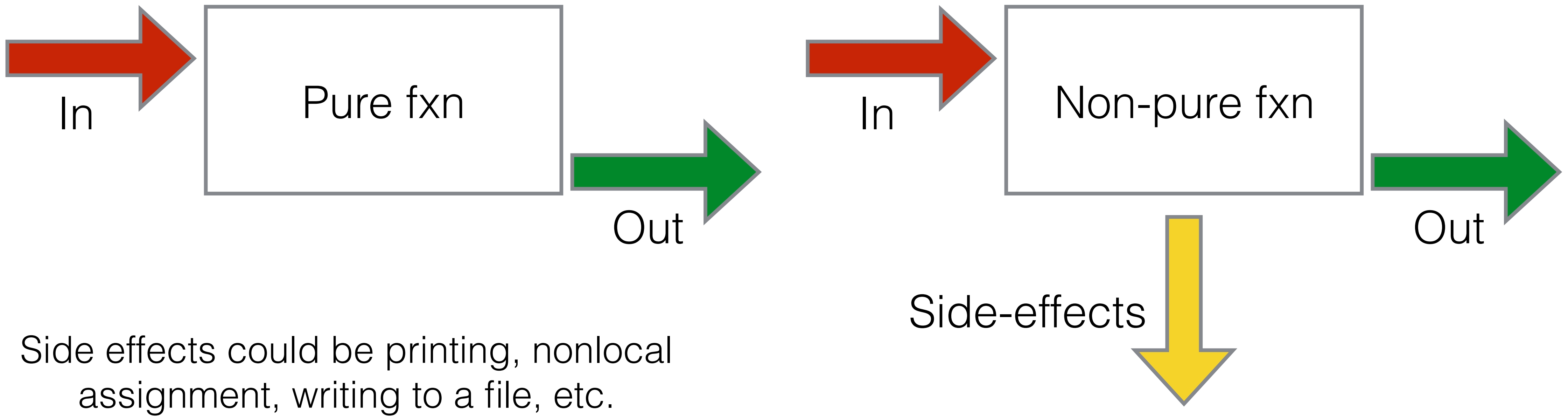
```python
def mdfy3(x):
    def inner():
        nonlocal x
        x = x + 1
    inner()
    return x
```

```
>>> x = mdfy3(20)
>>> x
21
```

Nonlocal

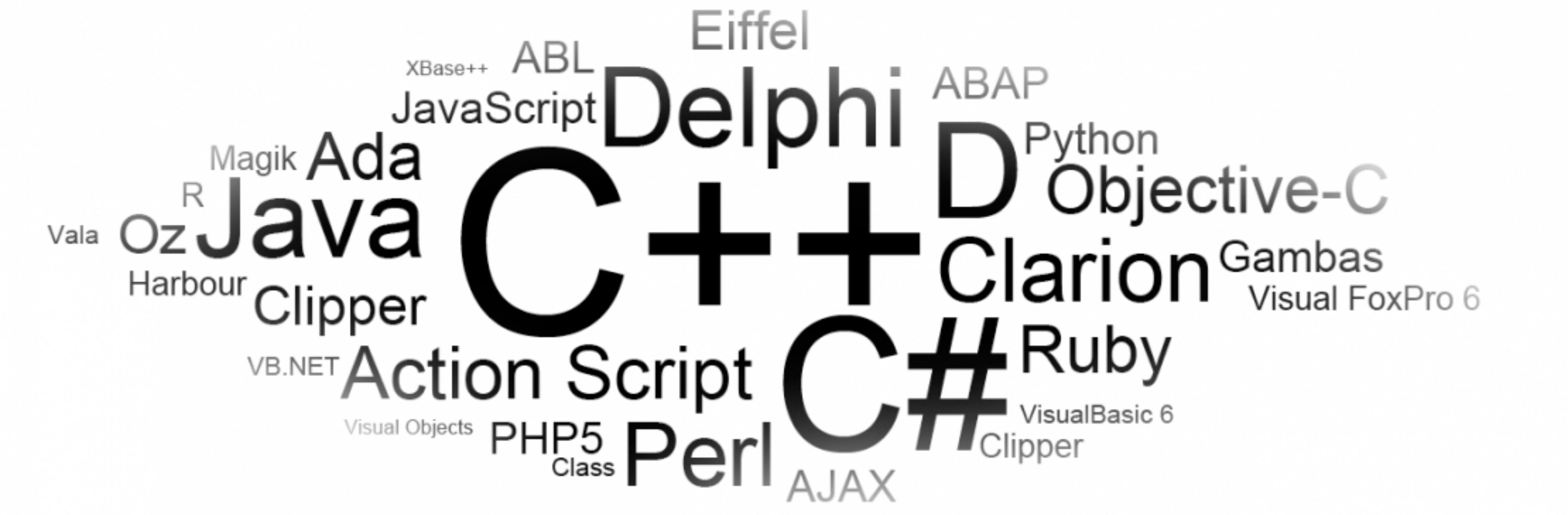
Exercise Caution:

- Nonlocal functions are non-pure
- As a reminder:



Side effects could be printing, nonlocal assignment, writing to a file, etc.

Object Oriented Programming



http://www.kamyacademy.com/wp-content/uploads/2014/01/object-orientated-programming-langs.png

Objects/Classes

Objects

- A (hopefully) more intuitive way of **representing data**
- **Common interface** means **powerful abstraction** (more on this later)

Objects/Classes

Classes

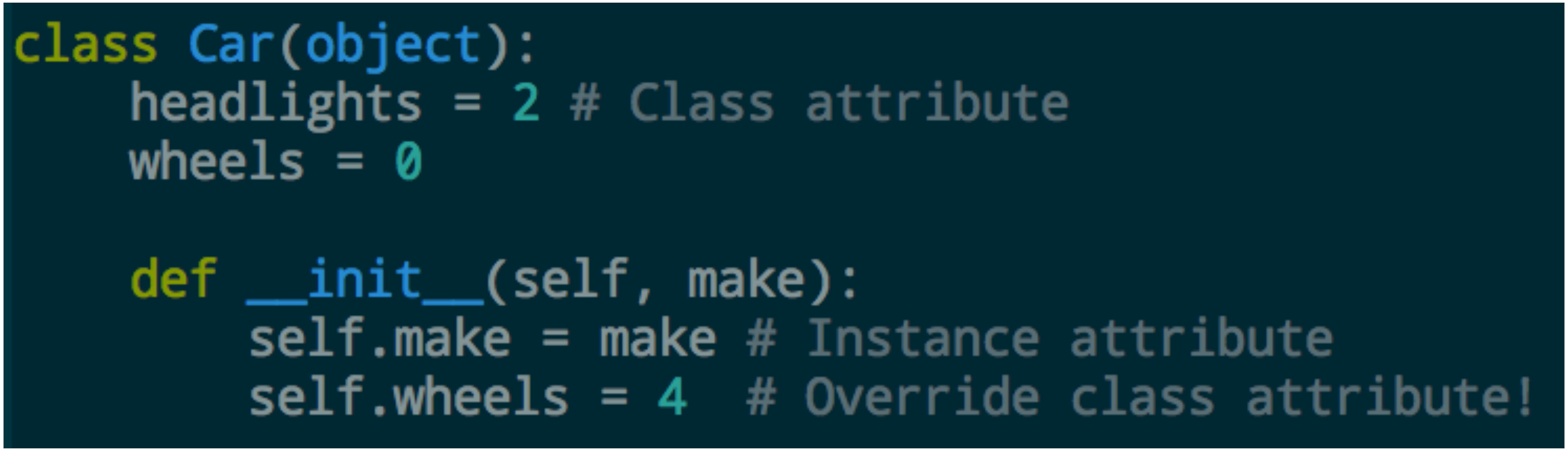
- A **“blueprint”**
- Objects are an **instance** of a class

http://velvetchainsaw.com/wp-content/uploads/2010/06/blueprint.jpg

Objects

- Attributes - **data!**
  - **Class attributes** is shared by the class
  - **Instance attributes** belong to an instance
- Methods - **behavior!**
  - Callable by instances

Attributes



```
class Car(object):
    headlights = 2 # Class attribute
    wheels = 0

    def __init__(self, make):
        self.make = make # Instance attribute
        self.wheels = 4  # Override class attribute!
```

Class vs Instance

Differences between **class** and **instance:**

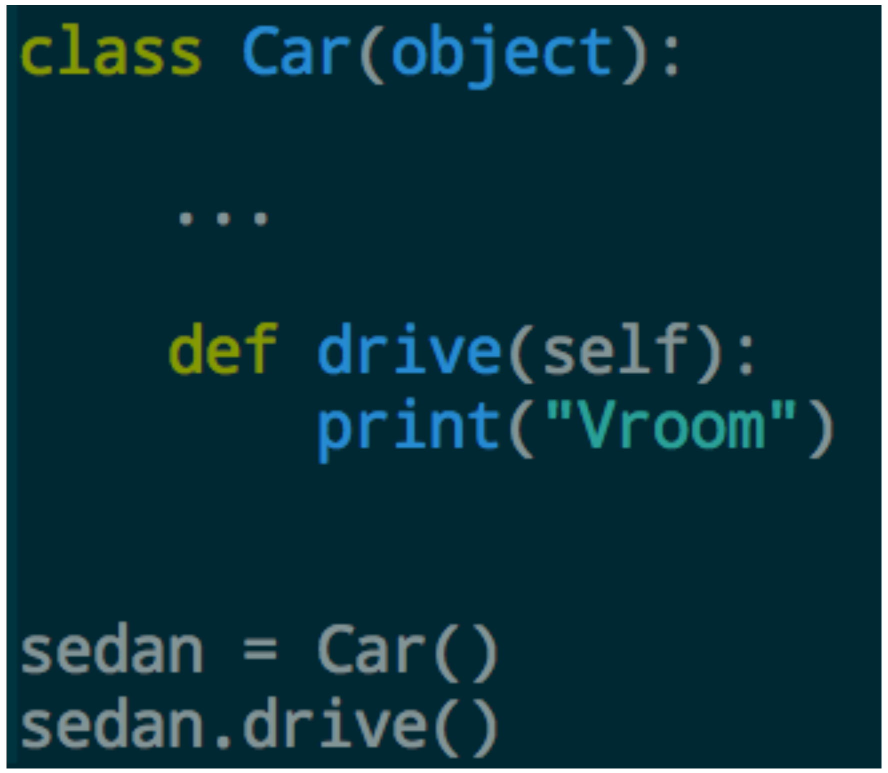
- Instance variables **take precedence** over class variables (instances are more specific than classes)
- However, new instance **defaults** to the class variables unless they are changed in the constructor (common) or somehow modified elsewhere.

Methods

Objects have a **bound method** associated with them

Dot expressions used to pass in an instance into "self"

This is implicitly "self"

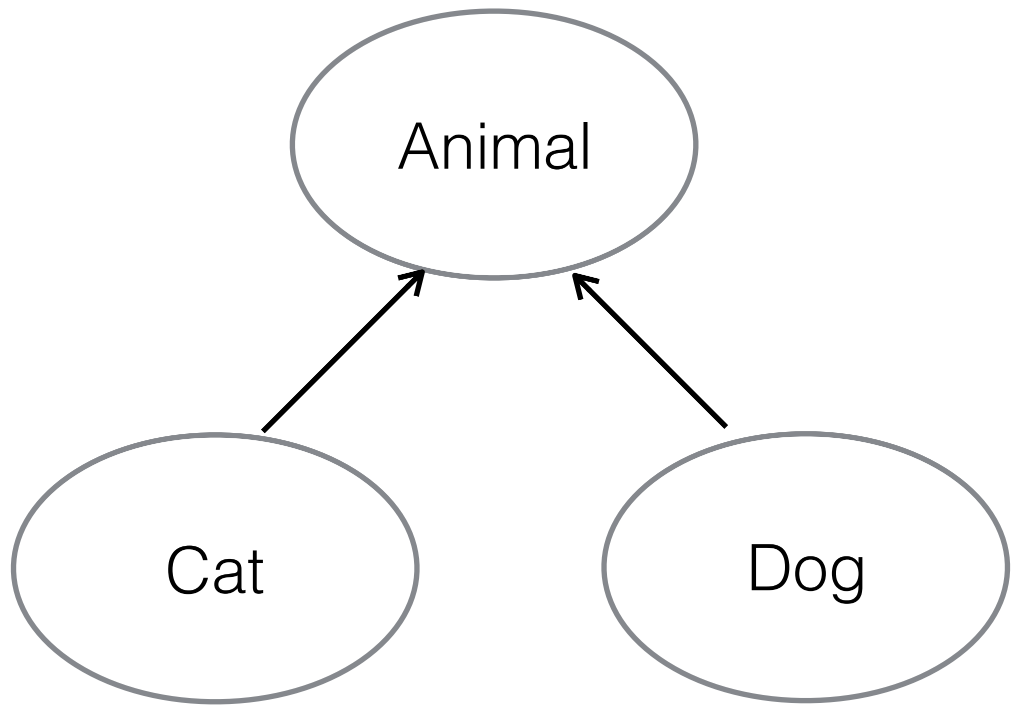


```
class Car(object):

    ...

    def drive(self):
        print("Vroom")


sedan = Car()
sedan.drive()
```

Inheritance

Write once, reuse forever

Reuse code by **applying "is-a" relationships**



Cat **is an** Animal and Dog **is an** Animal but Cat is not a Dog

Inheritance

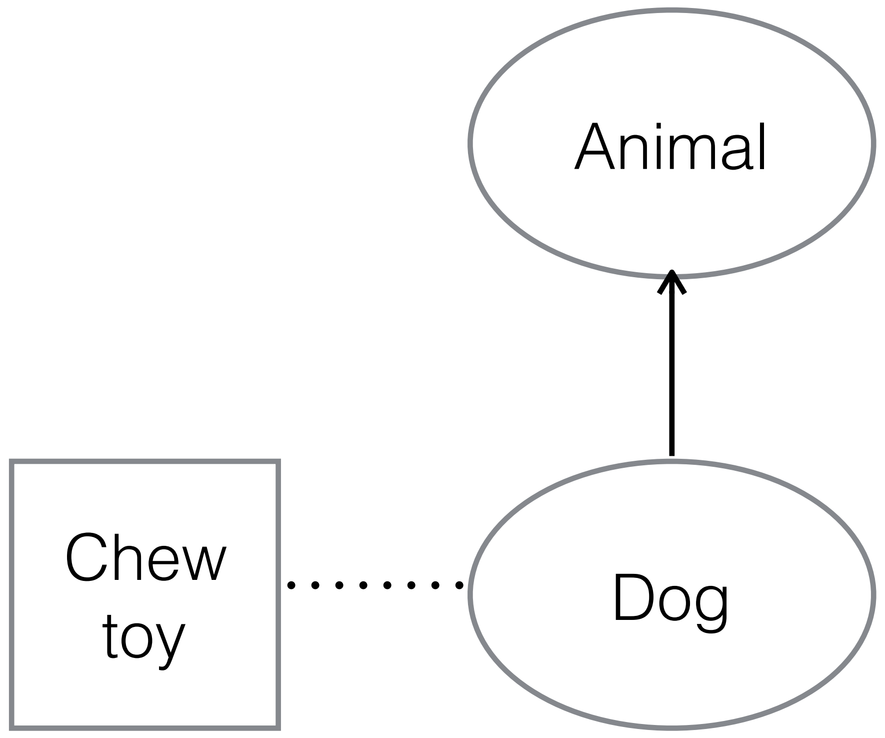
Can access/use **attributes** and **methods** from your parent class

- Don’t have to use them, can choose to **override**
- However, **parent’s behavior is present by default**

Inheritance

Beware: not everything should be inherited (“is-a”)!

Sometimes, composition or **“has-a”** relationships are better.



Dog **is an** Animal and **has a** chew toy.