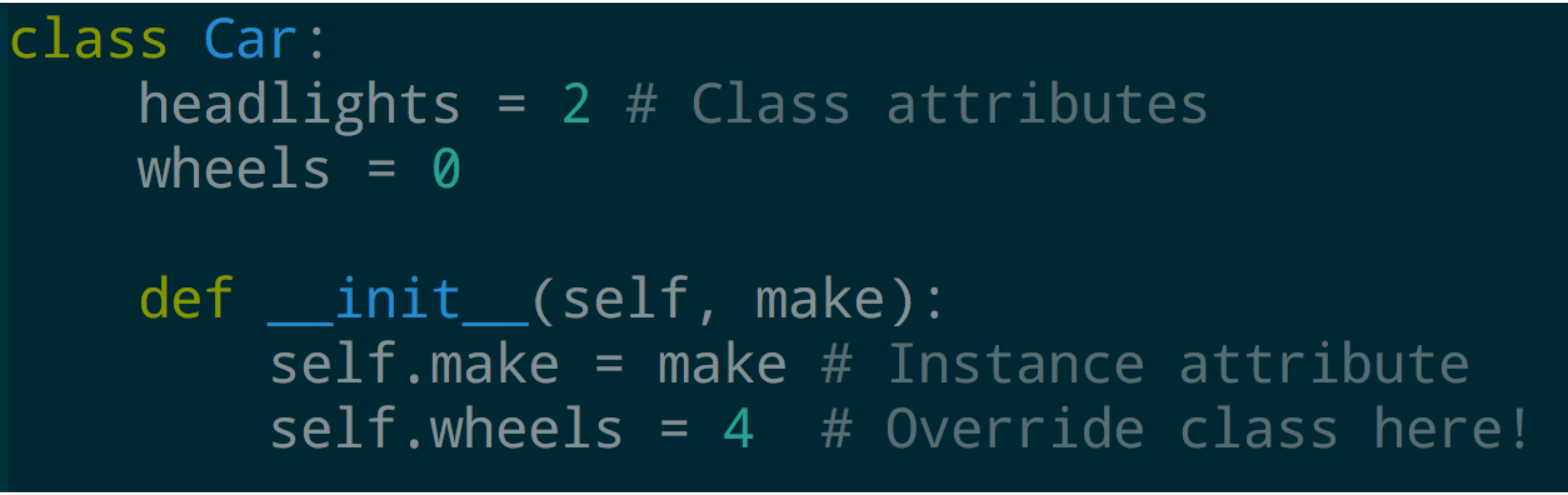
Discussion 05: **Mutation & Object Oriented Programming**

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

Vote A

```python
class Car:
    headlights = 2 # Class attributes
    wheels = 0

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

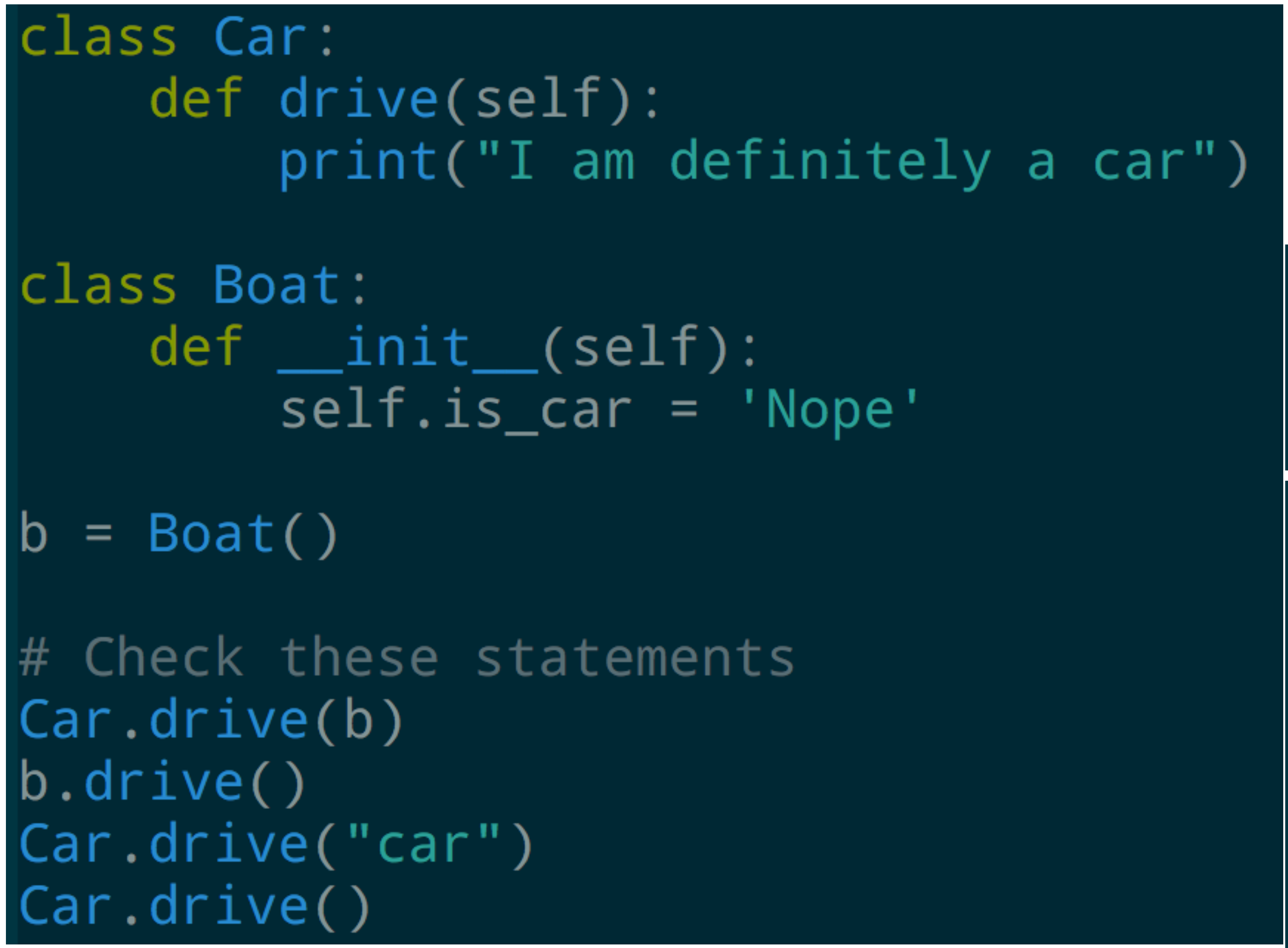
Vote B



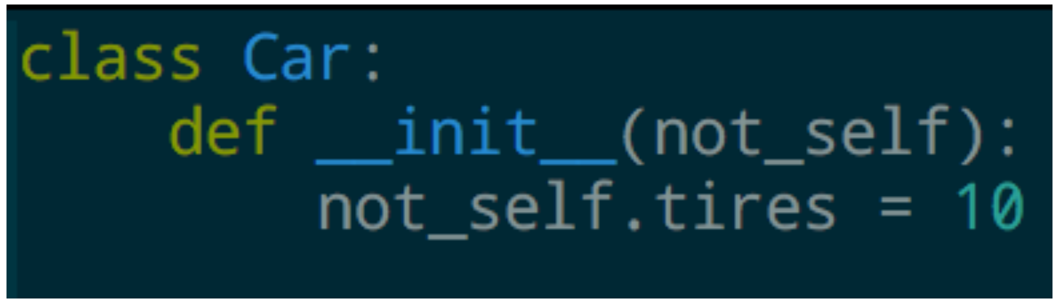
```python
class Car:
    headlights = 2 # Class attributes
    wheels = 0

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

Check Your Understanding

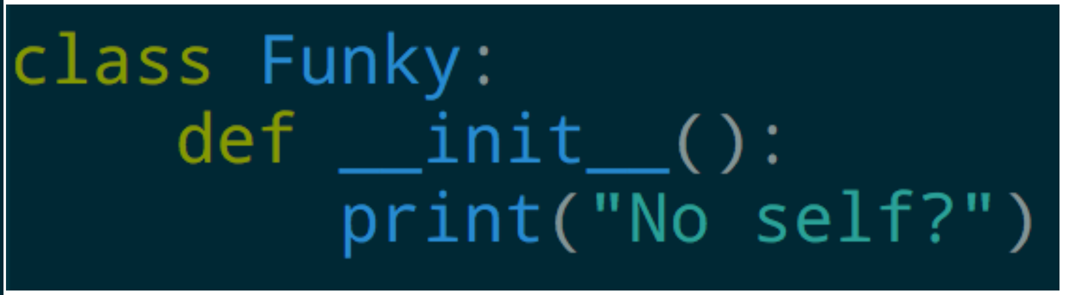
Which of the following are ok?



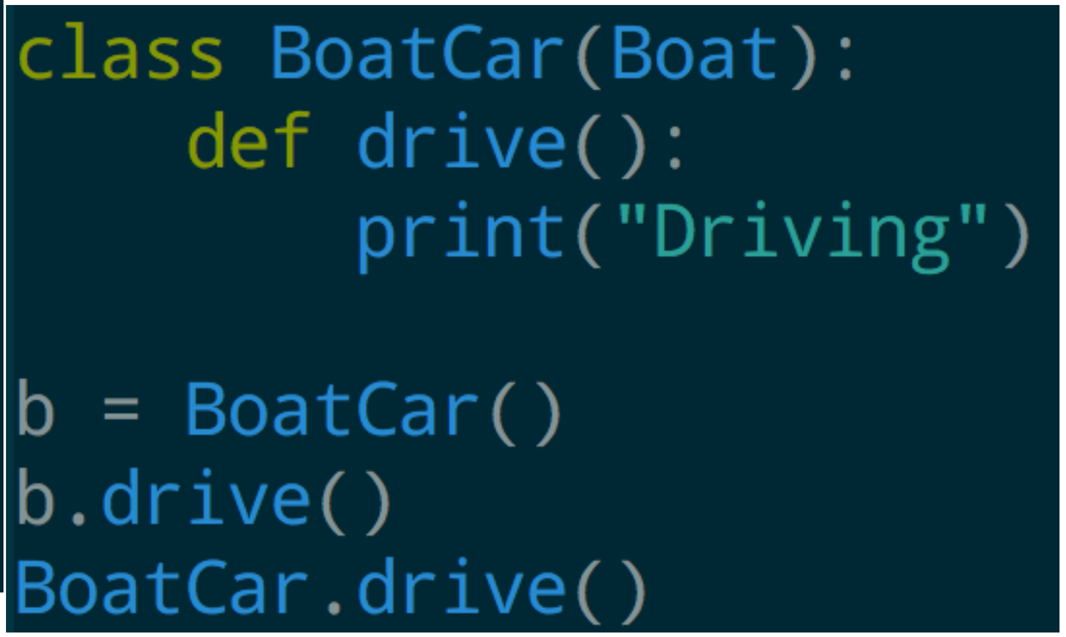
```python
class Car:
    def drive(self):
        print("I am definitely a car")

class Boat:
    def __init__(self):
        self.is_car = 'Nope'

b = Boat()

# Check these statements
Car.drive(b)
b.drive()
Car.drive("car")
Car.drive()
```



```python
class Car:
    def __init__(not_self):
        not_self.tires = 10
```



```python
class Funky:
    def __init__():
        print("No self?")
```



```python
class BoatCar(Boat):
    def drive():
        print("Driving")

b = BoatCar()
b.drive()
BoatCar.drive()
```

Agenda

1. List Mutation (questions)
2. OOP
3. Inheritance

Midterm Thoughts

"One test will not define who you are and whether or not you'll be a successful computer scientist."

Announcements

Maps due next Tuesday (bonus point for 1 day early)

Submit **Hog revisions** by 03/05

Submit **midterm regrade requests** by 03/05

Lab feedback **tiny.cc/jerrylabfb**

Discussion feedback **tiny.cc/jerrydiscfb**

The Complete List

Midterm scores released, submit regrades by March 5th.

Hog Project grades and composition scores emailed out to all students. Composition revisions due Sunday, March 5th.

Maps released due Tues 2/28, extra point Mon 2/27

- Proj party Thurs 2/23, Mon 2/27 6:30-8:30pm in 247 Cory.

Exam prep office hours. TA will give you time to work through some problems, give an overview of the relevant topics, and walkthrough the problems.

- See Piazza, this week Wed 4-6pm, Thurs 12-1pm, Sun 11-12pm.

Manas' discussion is faster paced, and you will get through all problems on the worksheet as well as the extra problems. Thurs 5-6:30 in 3105 Etcheverry. Attendance will count as normal, so you can attend this section instead of your normal section.

Labs 0-3, HW 1-3 and Hog all graded, please double check OK and email me if there are any problems.

Guerrilla Section Trees, Linked Lists and Mutable Values Sat 12-3pm in 247 Cory

No lecture Friday

Free One on One tutoring available. Sign up on Friday at 3:30 on Piazza.

List Mutation

"Static" lists are great, but…

- Having to copy information can be wasteful!
- Would like to modify our existing lists
- Downsides?

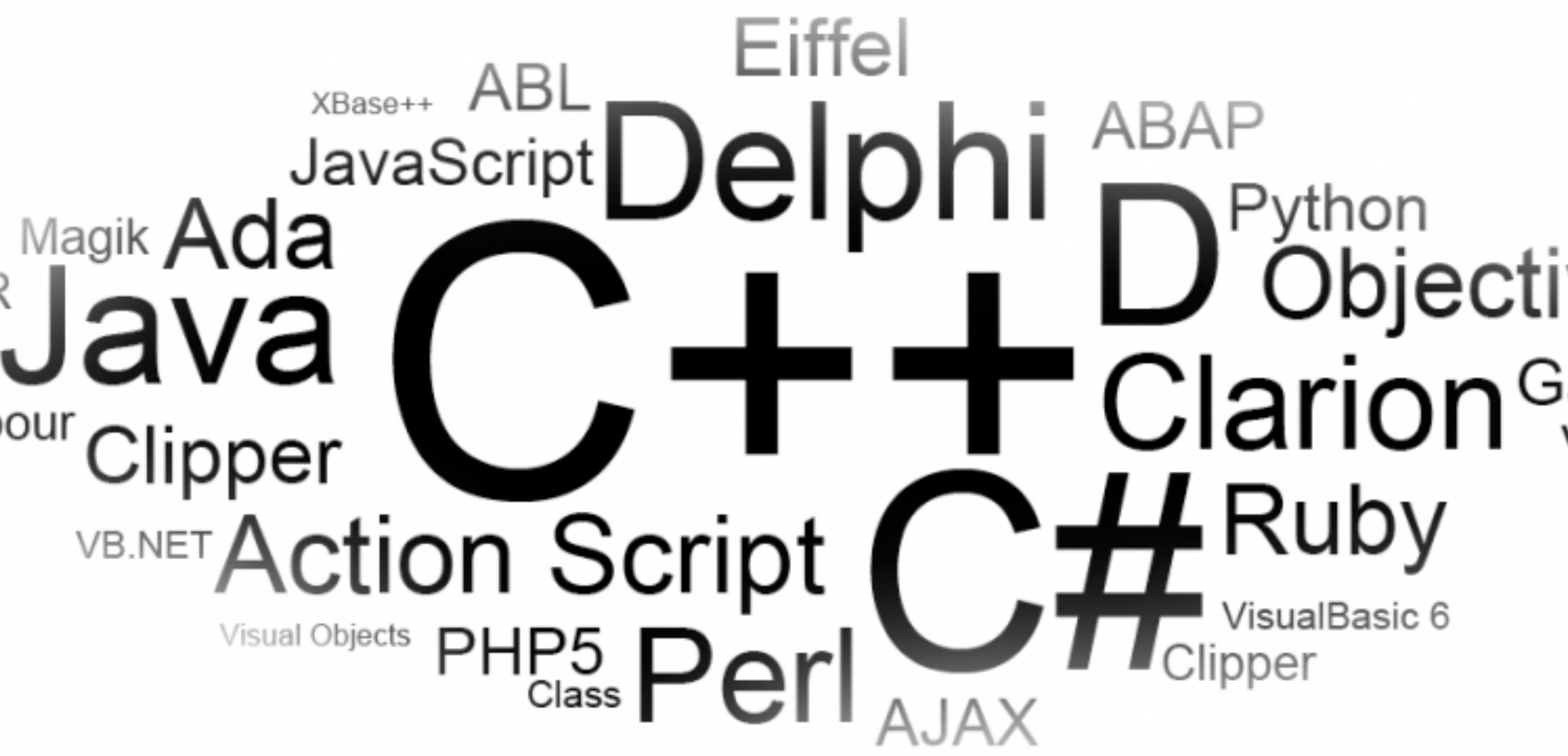
List Mutation

Operation	**Description**
`lst.append(x)`	Add x to the end of lst
`lst[1] = x`	Assign x to index 1
`lst = lst + [x]`	Append x to a **copy of** lst
`lst.remove(x)`	Remove first occurrence of x
`lst.pop(2)`	Remove **and** return element at index 2

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [1, 2, 3]
>>> lst1 == lst2 #compares each value
True
>>> lst1 is lst2 #compares references
False
>>> lst2 = lst1
>>> lst2 is lst1
True
>>> lst1.append(4)
>>> lst1
[1, 2, 3, 4]
>>> lst2
[1, 2, 3, 4]
```

```
>>> lst2[1] = 42
>>> lst2
[1, 42, 3, 4]
>>> lst1 = lst1 + [5]
>>> lst1 == lst2
False
>>> lst1
[1, 42, 3, 4, 5]
>>> lst2
[1, 42, 3, 4]
>>> lst2 is lst1
False
```

Object Oriented Programming



Objects/Classes

Objects

- A (hopefully) more intuitive way of **representing data**
- A commonly used method of **organizing a program**
- Formally split "global state" and "local state"

Classes

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

More Vocab

Attributes - **data**

- **Class attributes** is shared by the class
- **Instance attributes** belong to an instance

Methods - **behavior**

Class vs Instance

- **Instance attributes take precedence** over class attributes

- However, **new instance defaults to the class attributes** unless they are changed in the constructor or somehow modified elsewhere.

Attributes

```
class Car:
    headlights = 2 # Class attributes
    wheels = 0

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

__init__

```
def make_city(self, name, lat, lon):
    return [name, lat, lon]
```

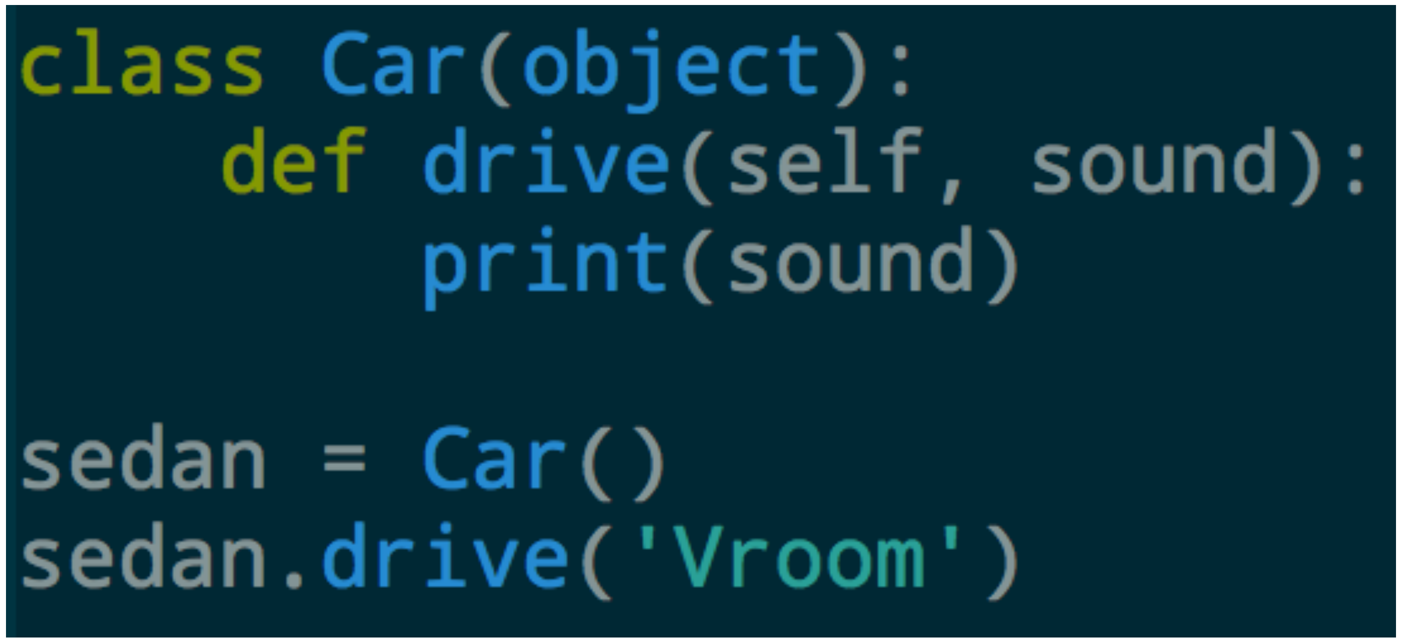
```
def __init__(self, name, lat, lon):
    self.name = name
    self.lat = lat
    self.lon = lon
```

Methods

A **bound method** combines a function and an instance

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

sedan is "implicitly self" ······▶

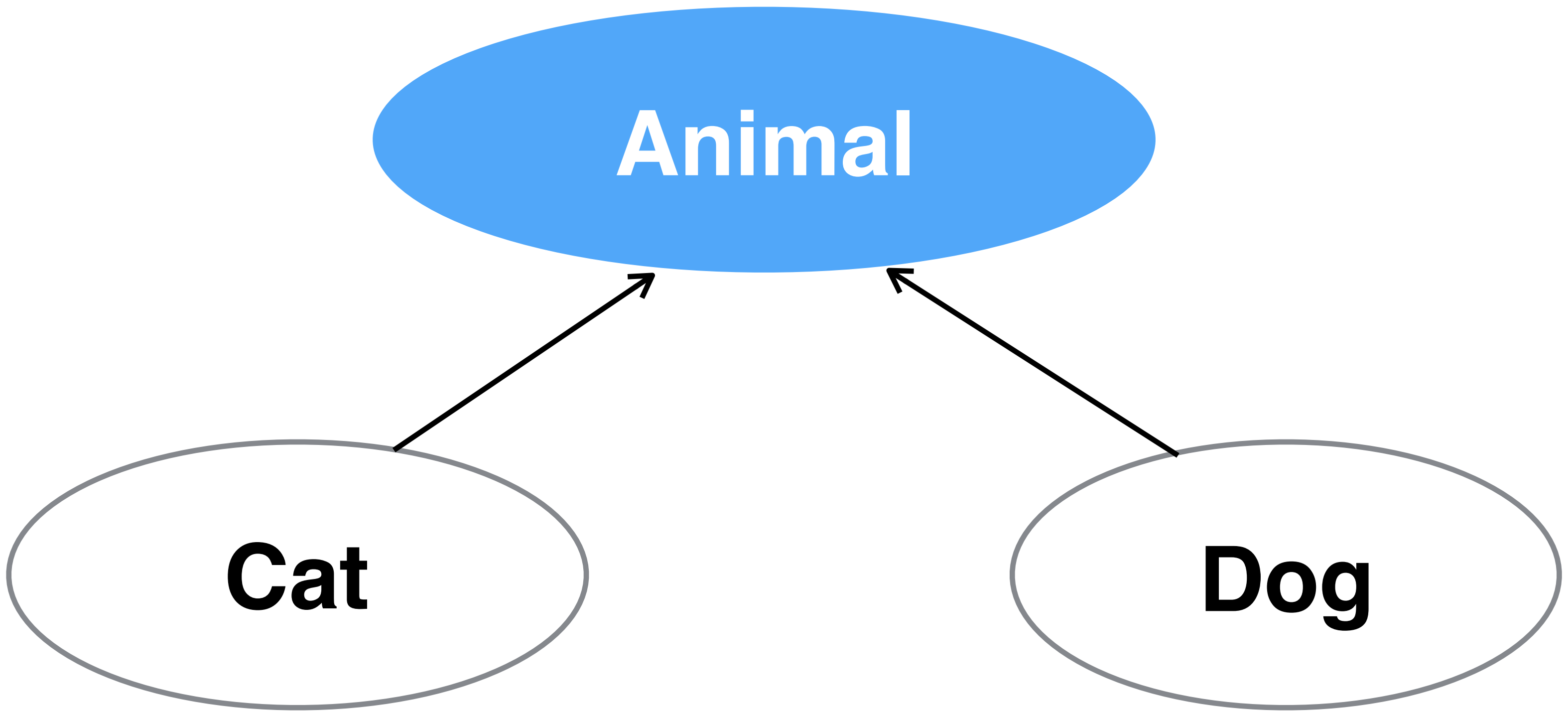


```
class Car(object):
    def drive(self, sound):
        print(sound)

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

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

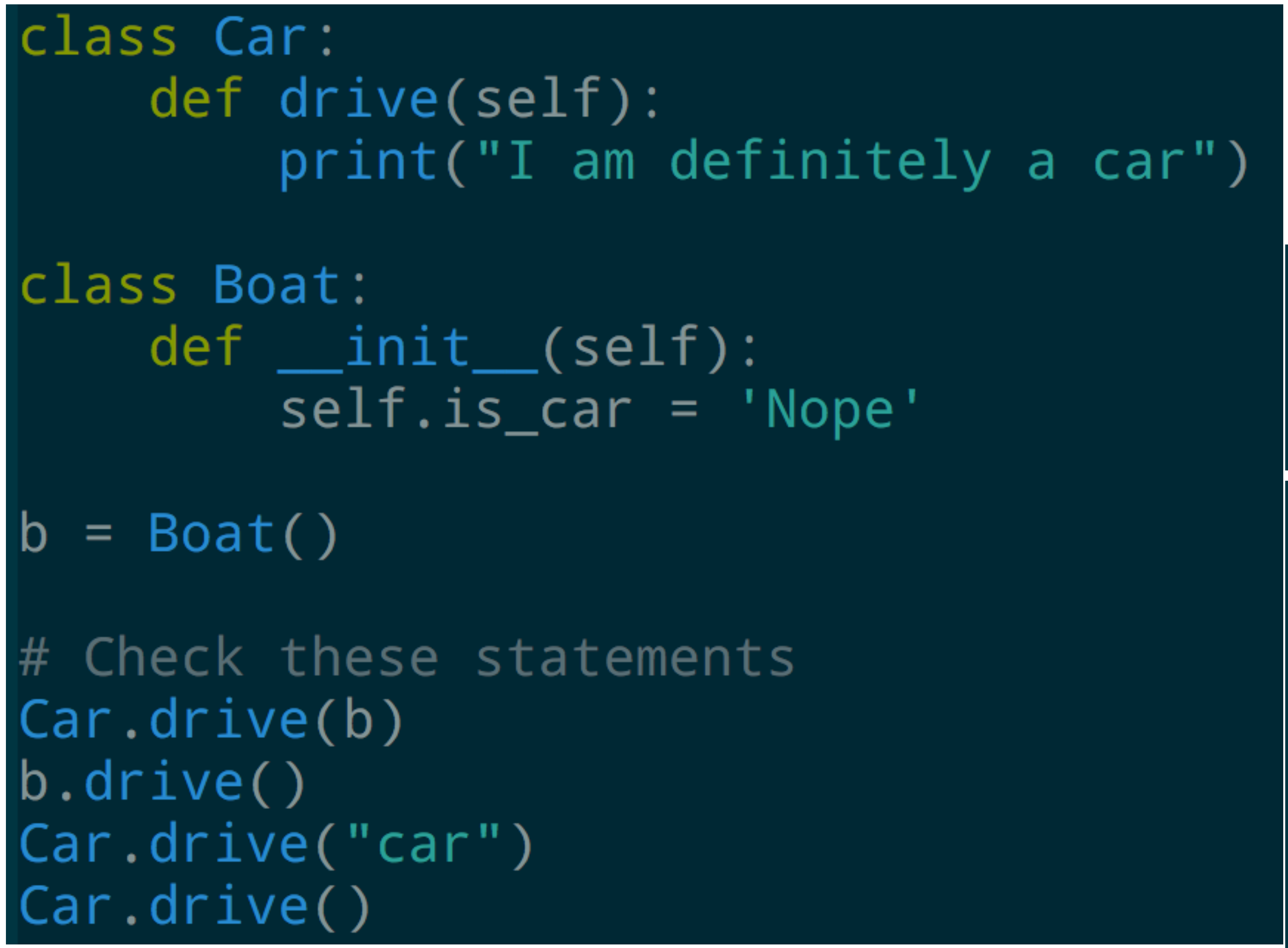
Inheritance

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

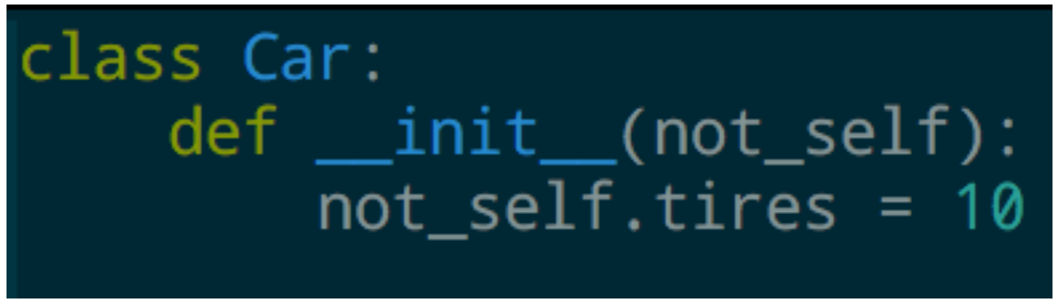
- Don’t have to use them, can choose to **override**
- However, **parent’s behavior is present by default**

Check Your Understanding

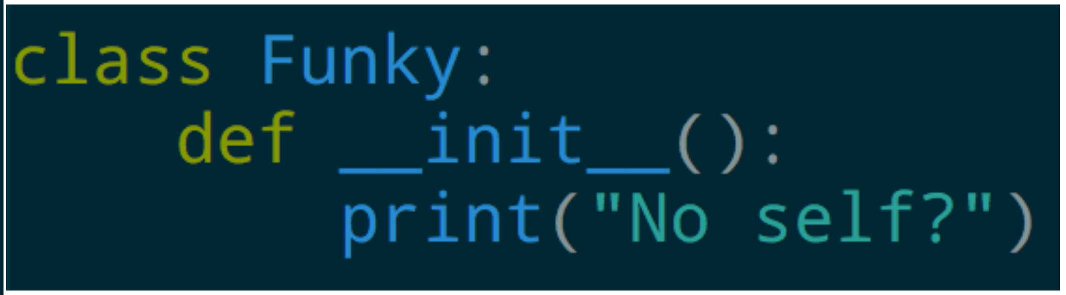
Which of the following are ok?



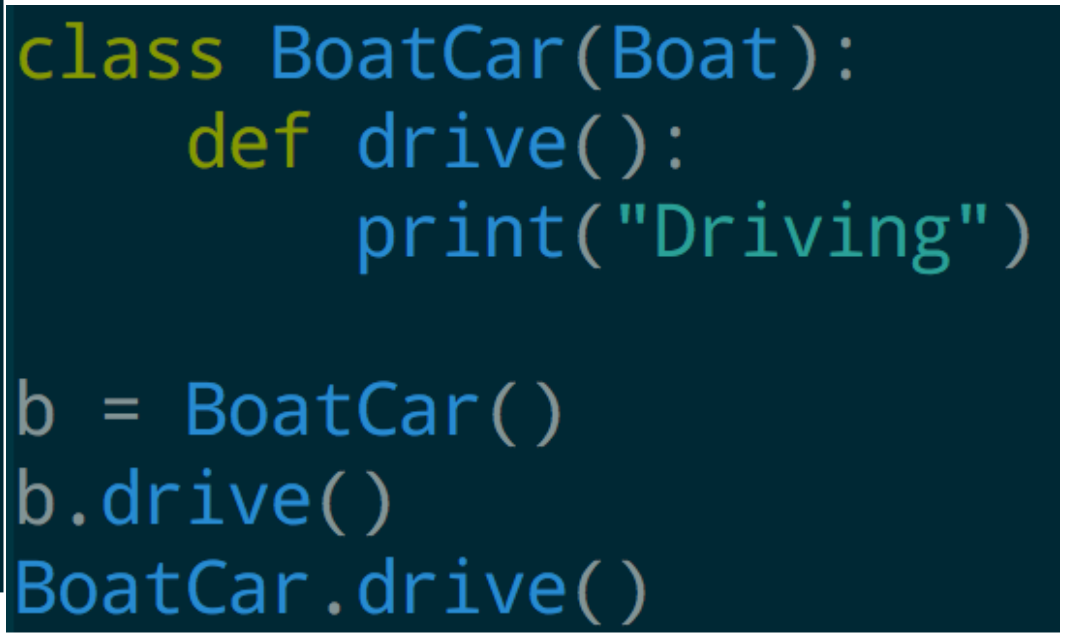
```python
class Car:
    def drive(self):
        print("I am definitely a car")

class Boat:
    def __init__(self):
        self.is_car = 'Nope'

b = Boat()

# Check these statements
Car.drive(b)
b.drive()
Car.drive("car")
Car.drive()
```



```python
class Car:
    def __init__(not_self):
        not_self.tires = 10
```



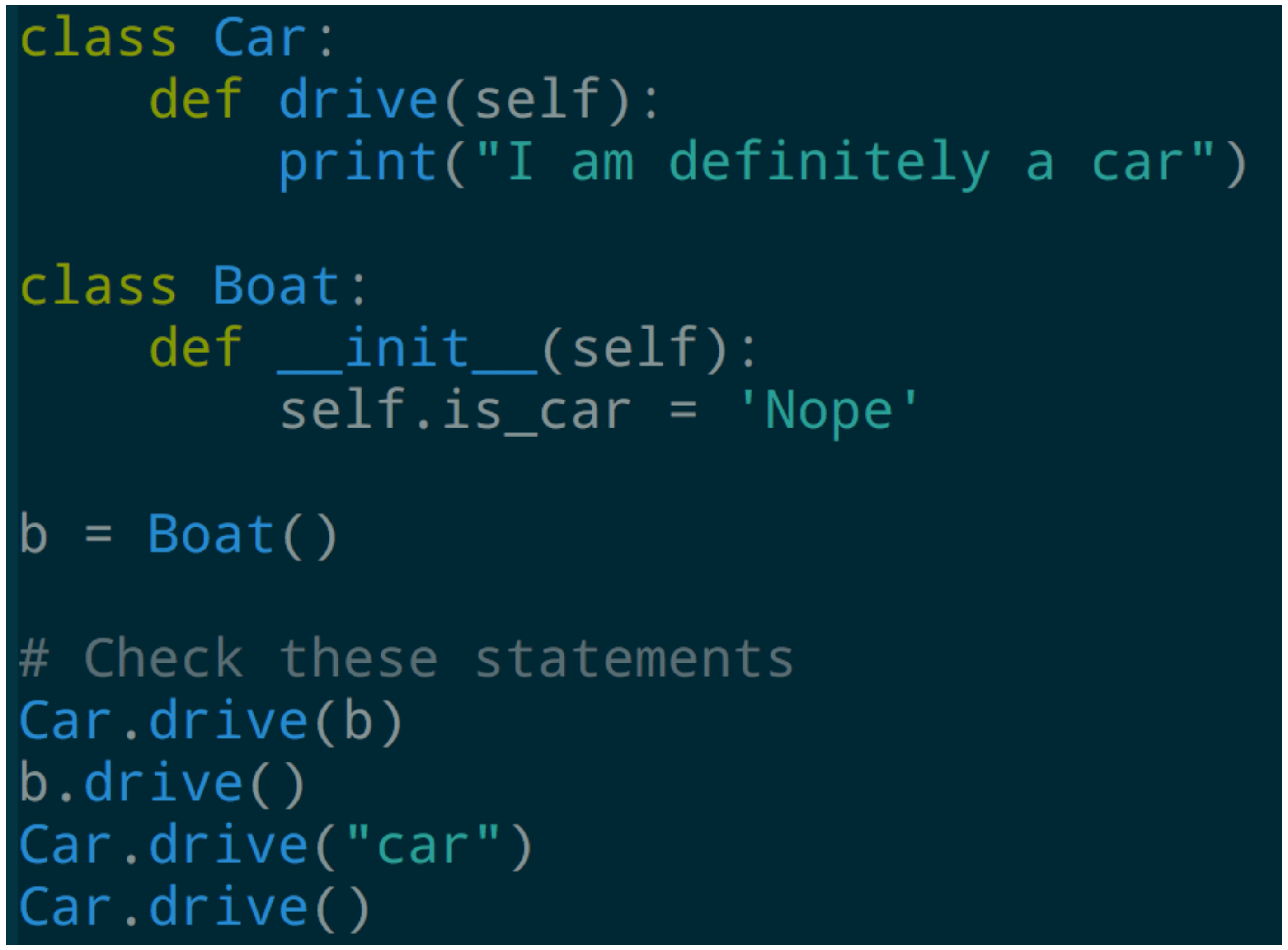
```python
class Funky:
    def __init__():
        print("No self?")
```



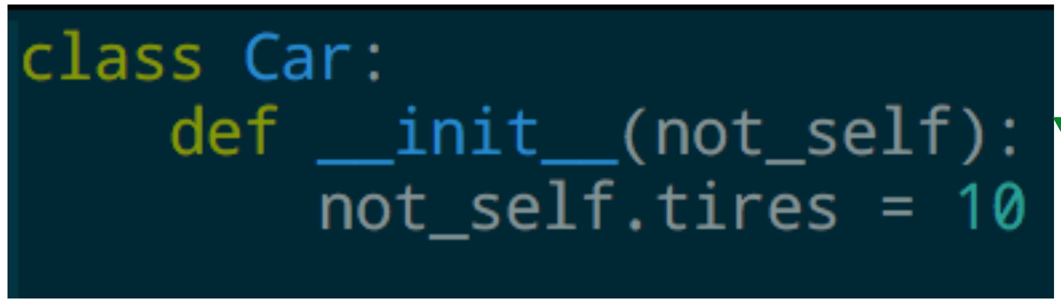
```python
class BoatCar(Boat):
    def drive():
        print("Driving")

b = BoatCar()
b.drive()
BoatCar.drive()
```

Odds & Ends

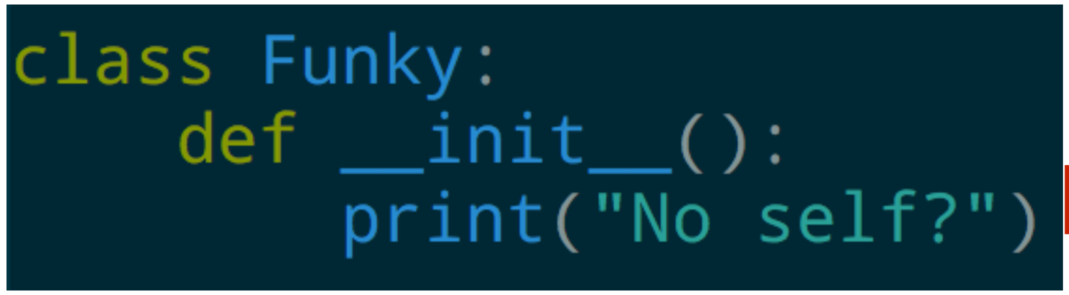
Which of the following are ok?



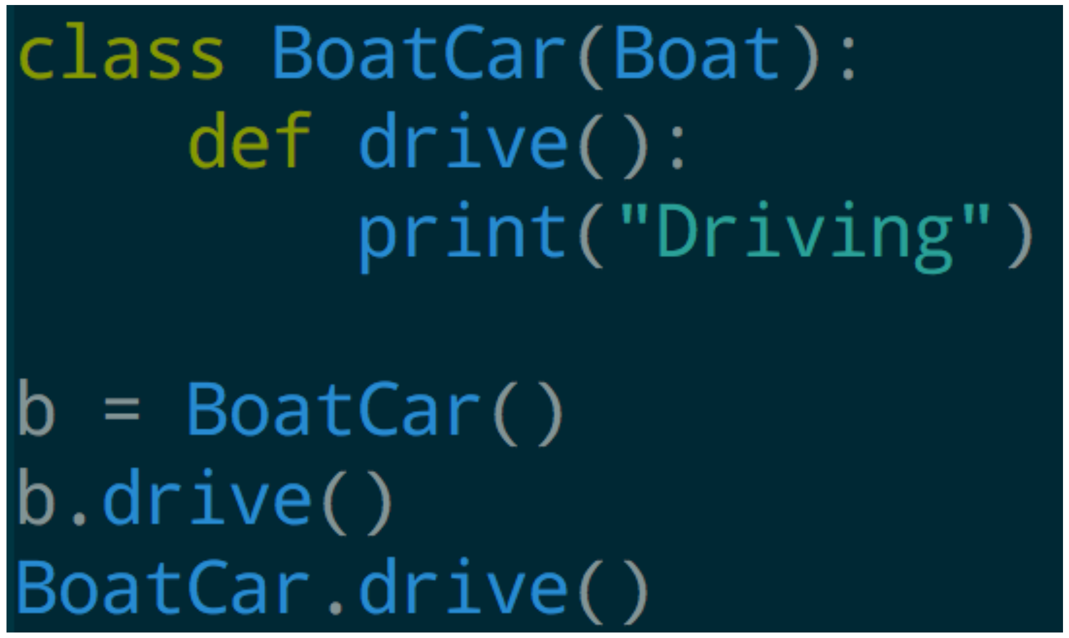
```
  class Car:
      def drive(self):
          print("I am definitely a car")

  class Boat:
      def __init__(self):
          self.is_car = 'Nope'

  b = Boat()

  # Check these statements
Y Car.drive(b)
N b.drive()
Y Car.drive("car")
N Car.drive()
```



```
class Car:
    def __init__(not_self):
        not_self.tires = 10     Y
```



```
class Funky:
    def __init__():
        print("No self?")       N
```



```
  class BoatCar(Boat):
      def drive():
          print("Driving")

  b = BoatCar()
N b.drive()
Y BoatCar.drive()
```