Welcome to 61A Lab!

We will begin at **5:10**!

Slides: **cs61a.bencuan.me**

Announcements

- HW4 is due this Thursday!
- Ants is released!
  - Checkpoint 1 due this Thursday also
    - Rest assured, no bees were harmed in the making of this project

The Plan

- OOP syntax review
- Inheritance
- Work time!

Object Oriented Programming

Discussion review

- **Class:** a blueprint for making objects
- **Instance:** one of those objects

- Class attribute: shared by **all** objects of that type (# wheels, model)
- Instance Variable: specific to **your** object (gas, mileage)

Syntax

```
class Car:    ← blueprint for all car objects
    num_wheels = 4   ← class attributes: shared by all Cars

    def __init__(self, color):
        self.wheels = Car.num_wheels   ← instance attributes: each car has its own values
        self.color = color

    def drive(self):   ← class method (aka function): takes a Car instance as self
        if self.wheels <= Car.num_wheels:
            return self.color + ' car cannot drive!'
        return self.color + ' car goes vroom!'

    def pop_tire(self):
        if self.wheels > 0:   ← remember to put self in front of instance attributes!
            self.wheels -= 1
```

OOP Demo!

This example is also on the top of the lab page.

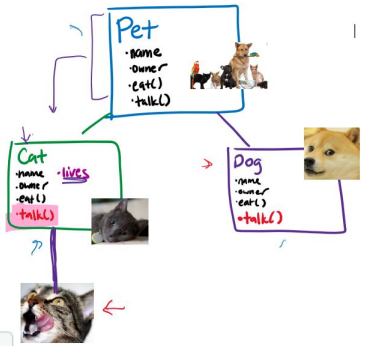
pythontutor link

Inheritance

Inheritance

- When a more specific type of object (child) **inherits** behaviors from a general type (parent)
- Example: 61A (parent) **is a** CS class (child)
- Access parent information using `super()`
  - `note the parentheses!!!!!`
- Methods that are not **overridden** (defs with same name) are automatically taken from parent

Inheritance Example



Before inheritance:

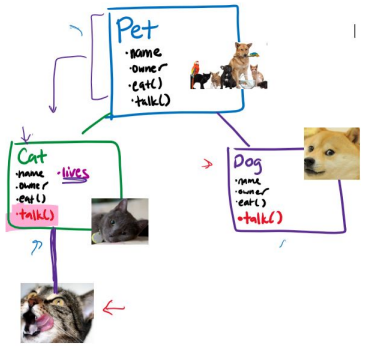
```
class Dog():
    def __init__(self, name, owner):
        self.is_alive = True
        self.name = name
        self.owner = owner
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name + " says woof!")

class Cat():
    def __init__(self, name, owner, lives=9):
        self.is_alive = True
        self.name = name
        self.owner = owner
        self.lives = lives
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name + " says meow!")
```

lots of duplicate code!

and what if we wanted to add more animals that also do basically the same thing?

Inheritance Example



With inheritance:

```
class Pet():
    def __init__(self, name, owner):
        self.is_alive = True    # It's alive!!!
        self.name = name
        self.owner = owner
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name)

class Dog(Pet):
    def talk(self):
        print(self.name + ' says woof!')
```

simple to extend, less code needed!

Lab Q2 task: extend Cat one more time using the same ideas (calling super(), reuse code as much as possible)

```
class Cat(Pet):
    def __init__(self, name, owner, lives=9)
        super().__init__(name, owner)
        self.lives = 9

    def talk(self):
        print(self.name + ‘ says meow!’)
```

Lab Hints

Lab Hints

- Remember to use self. when accessing instance variables!
- Read the doctests very carefully! they will tell you the desired behavior
- Reuse variables and methods from parent classes as much as possible! use super().
- Remember the difference between class vs instance variables: make sure you know when you’re changing something for all objects vs. just one

Work Time!

go.cs61a.org/ben-queue