# Discussion 07

## Object Oriented Programming, Representation

Aditya Balasubramanian
aditbala [at] berkeley [dot] edu

Slides available at `teaching.aditbala.com`

# Announcements 📣

- Homework 5 is due Thursday 10/13
- Ants is due Friday 10/21.
  - Phase 1 checkpoint Friday 10/14.
  - Phase 2 checkpoint Tuesday 10/18.
  - Early submission bonus Thursday 10/20.
- Project parties 5pm-7pm on Wed 10/12, Mon 10/17, & Wed 10/19.

# Object Oriented Programming (OOP) 

# Object Oriented Programming (OOP)

- What is OOP?
  - Use of classes to define our own data types
    - More abstraction
  - Reuse code with inheritance
    - MORE ABSTRACTION
- Those with prior experience in Java are familiar
- You have already used OOP!
  - `list.append`
  - `append` is a method belonging to the `list` class

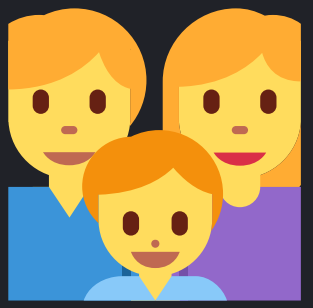
# Some Terminology

- Class
  - Template for creation of object
- Object
  - An instance of a class
- Variables
  - Instance Variables
    - property specific to an object
  - Class Variables
    - property shared between all instances of a class
- Method
  - Function that is bound to a class

# Functions vs Methods

- Methods need to take in `self` as an object
- `self` argument tells which object to call method on
- Two methods of writing method calls
  - `Class.method(self, args)`
  - `object.method(args)`
    - self is automatically set as object
- Demo

# Worksheet

# Inheritance 

```
class Dog():
    def __init__(self, name, owner):
        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.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!")
```

Credit to @Anto and @Poggenkemper

# What's the problem?

- Too much repeated code
- How to avoid this problem?
- Inheritance!

```
class Dog(Pet): # Dog inherits the Pet class - as in, all Dogs are Pets
    ...
```

Credit to @Anto

# Now with Inheritance!

```
class Pet():
    def __init__(self, name, owner):
        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): # Inherits all methods/variables from the Pet class
    def talk(self):
        print(self.name + ' says woof!')
```

Credit to @Anto

# Inheritance - super()

- `super()` will refer to methods in the parent class

```
class Cat(Pet): # Inherits all methods/variables from the Pet class
    def __init__(self, name, owner, lives = 9):
        super().__init__(name, owner)
        # same as calling Pet.__init__(self, name, owner) from here
        self.lives = 9
    def talk(self):
        print(self.name + ' says meow!')
```

Credit to @Anto

# Worksheet

# Representation

- `__str__`
  - return's a human readable form of object
- `__repr__`
  - return's a human readable form of object

# Representation Demo

```
class Rational:

    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

    def __str__(self):
        return f'{self.numerator}/{self.denominator}'

    def __repr__(self):
        return f'Rational({self.numerator},{self.denominator})'

>>> a = Rational(1, 2)
>>> str(a)
'1/2'
>>> repr(a)
'Rational(1,2)'
>>> print(a)
1/2
>>> a
Rational(1,2)
```

# Thank you!

**Anon Feedback -> https://tinyurl.com/adit-anon**