Discussion 6

OOP

Antonio Kam

anto [at] berkeley [dot] edu

Announcements

- Ants gets released very soon!
  - My favorite project by far
- HW 04 on Iterators/Generators
  - Problems are kinda hard - Office Hours can be pretty useful for this

Notes from last section

- favorite sport
  - basketball 🏀
- how much did u score for ur mt1
  - not relevant 💀
- mental health recovery methods after midterms
  - will talk more about this later
- ginger ale is the superior soda
  - i think i've never tried?

Notes from last section

- last discussion was good
  - 🥹
- Can you sing us a song?
  - might do it for the last discussion (i'd only really want to do this if i have my piano, so i'll have to bring it over on the last day probably)
- It’s ok if you don’t like Dr Pepper. In my opinion, it will always stay the best drink in the world
  - 💀
- Entity Cramming in Minecraft
  - I've seen some crazy stuff been done with entity cramming
  - Minecraft + CS 61C

Midterm

- Important to mention that the midterm is worth `64` points out of the total `300`
- Exams are not the only component of your grade in this class - you have discussion/lab attendance, and homeworks/projects where you're given unlimited attempts, and there are also no hidden tests
  - This means that quite a large portion of the points in this course are not based on exam performance.
- Many people struggle on exams; it's completely normal to not feel too good about your own performance (in fact, quite a lot of exams in higher education will have averages lower than what you may have been used to in HS)

Temperature Check 🌡️

- OOP
- Inheritance

All slides can be found on

`teaching.rouxl.es`

Object-Oriented Programming

What is OOP?

- One way I like to think of OOP is as a sort of 'advanced' data abstraction
  - You would use OOP for similar things that you would use data abstractions for
  - Can make a `City` class, for example
- OOP also allows for inheritance (less repetition of code, more on this later/next discussion)
- OOP also allows for mutation
  - Similar to list mutation (`.append`, `.extend`, etc.)
- You may have seen this if you've seen Java before (I didn't have any exposure to OOP when I took CS 61A for the first time)

OOP Terminology

- Class
  - A class is sort of a 'blueprint' for something. You can think of it as a template for creating an object
- Instance
  - An instance of a class is one object of that blueprint, or one physical object that you create based on your template
- Variables
  - Instance Variables: Variables unique to each instance (each actual object)
  - Class Variables: Variables shared between each instance in the same class
- Method
  - Function bound to a class

Functions vs Methods

- Methods need to take in `self` as an argument
  - This is very often *implicitly* passed in when the thing on the left side of the dot is an instance
  - `my_car.drive(100)` is the same as `Car.drive(my_car, 100)`
- `self` refers to the actual instance (rather than the class)
- Two ways of calling methods:
  - `Class.method(self, args)`
  - `instance.method(args)`

Q1 Mini-lecture

Worksheet!

Inheritance

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!")
```

Inheritance

Notice the redundancies in the code? One of the core foundations in this class is to not repeat yourself (DRY)

- Instead, you can use inheritance to solve this problem
- Syntax when creating a class is to put brackets around the class you want to inherit:

```
class Cat(Pet): # Cat inherits the Pet class - as in, all cats are pets (in this world)
    ...
```

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

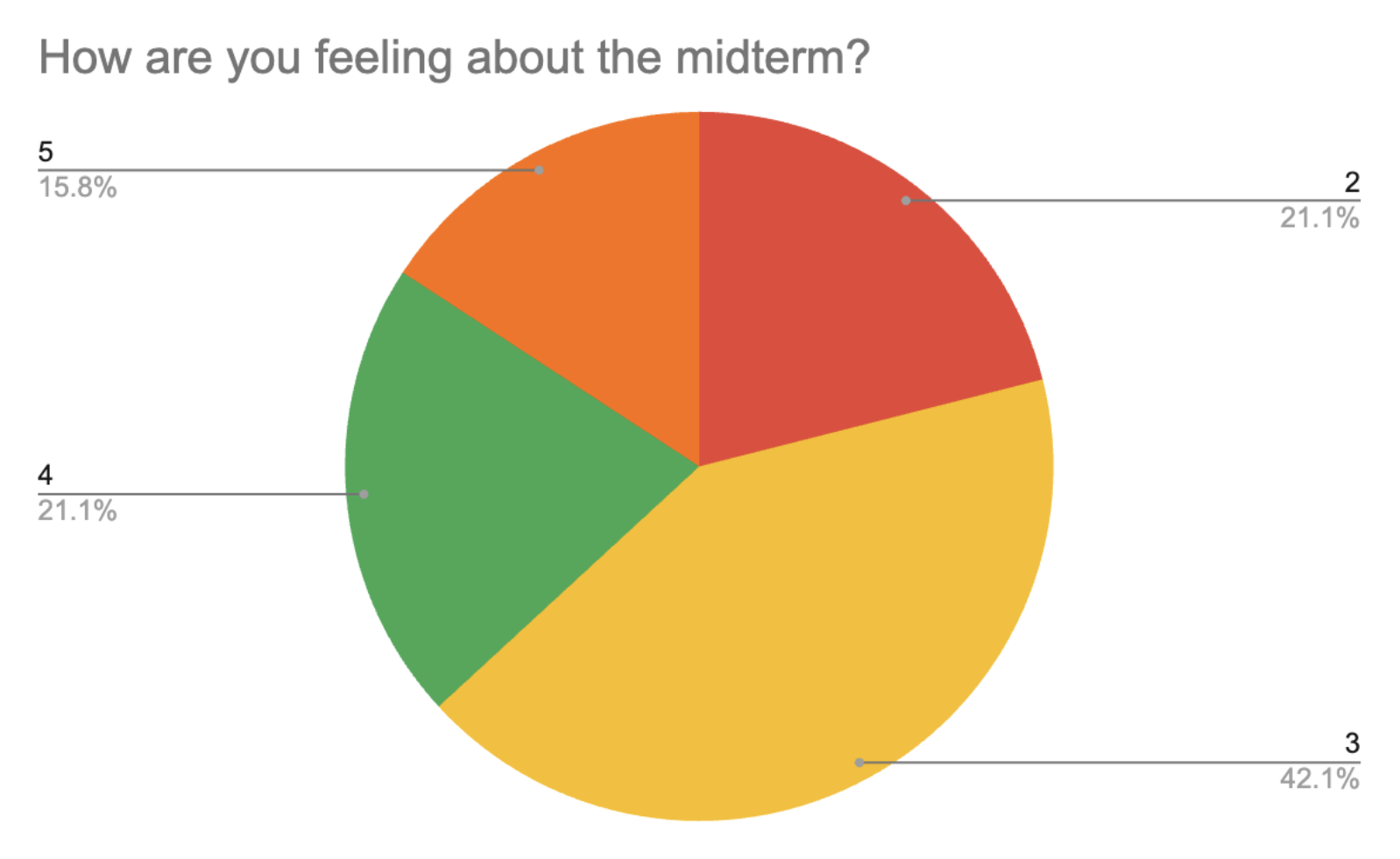
Inheritance - `super()`

- Calling `super()` will refer to the class's *superclass*
- You can use the parent's method and then add on to that.

```
class Cat(Pet): # Inherits all methods/variables from the Animal 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!')
```

Worksheet!

Results from last section (`links.rouxl.es/disc`)



Worksheet!

Mental Health Resources

- CAPS:
  - If you need to talk to a professional, please call CAPS at 510-642-9494.
- After Hours Assistance
  - For any assistance after hours, details on what to do can be found at this link

Anonymous Feedback Form

`links.rouxl.es/feedback`

Thanks for coming! 🥳

Please give me feedback on what to improve!