

LAB 08




OOP, Inheritance

Mingxiao Wei

mingxiaowei@berkeley.edu

Mar 14, 2023

LOGISTICS

- Lab 08 due tomorrow 03/15
 - PLEASE SUBMIT TWO FILES: `lab08.py` and `classes.py`
 - [More instructions here](#)
- ANTS is released!  
 - Checkpoint 1 due Fri 03/17
 - Checkpoint 2 due next Tue 03/21
 - The whole project due next Fri 03/24
 - Submit by next Thu 03/23 for one extra point!
- Homework 06 due this Thu 03/16
- All OH online today in anticipation of the rain 
- CLASS METHOD NO LONGER IN SCOPE **!!**

FROM LAST TIME... 👁️👁️

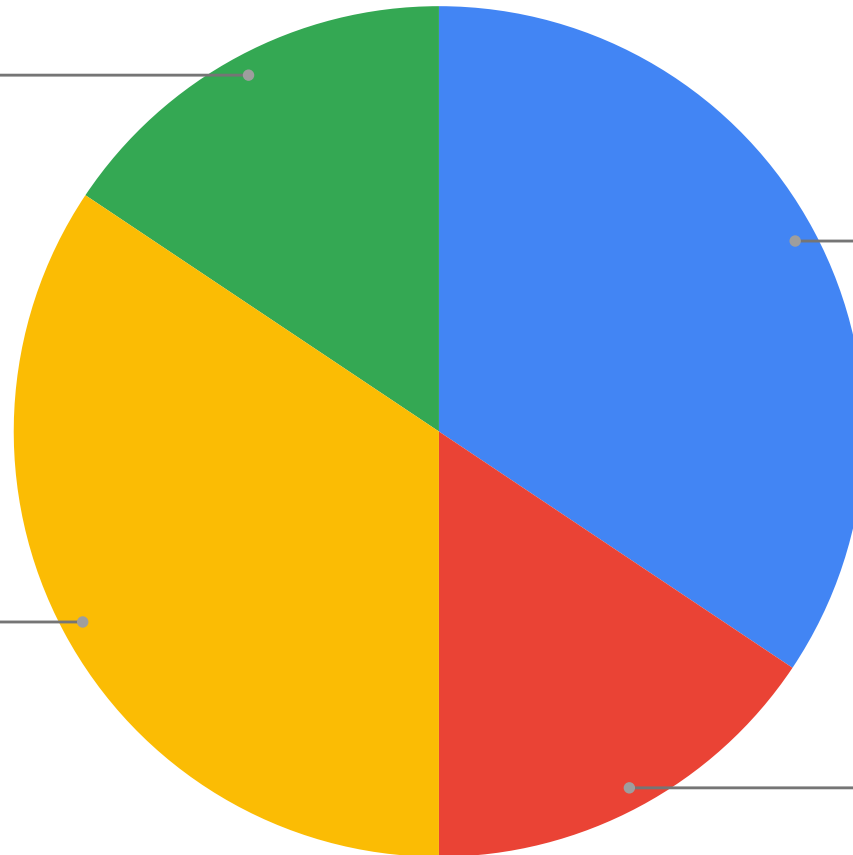
If you had the chance, would you rather...

Go back to high school
15.6%

Go back to elementary school
34.4%

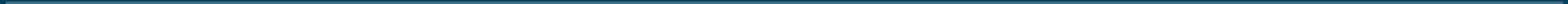
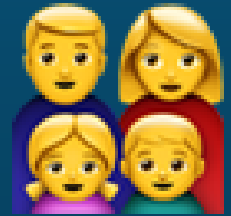
Stay where you are at
34.4%

Forward to the future
15.6%



Such a symmetric distribution 😲

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

- `Dog` and `Cat` have a lot in common - repeated code :(
- Solution - a base class, `Pet`, from which both classes inherit
 - DRY - Don't repeat yourself

superclass

class Pet	
is_alive	<code>__init__(self, name, owner)</code>
name	<code>eat(self, thing)</code>
owner	<code>talk(self)</code>



class Dog	
is_alive	<code>__init__(self, name, owner)</code>
name	<code>eat(self, thing)</code>
owner	<code>talk(self)</code>

class Cat	
is_alive	<code>__init__(self, name, owner)</code>
name	<code>eat(self, thing)</code>
owner	<code>talk(self)</code>
lives	

subclass

* super class and base class are used interchangeably

INHERITANCE

```
class Pet: # base class
    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): # A dog is a pet!
    def talk(self): # overridden bc it's different from the base class
        print(self.name + ' says woof!')
```

INHERITANCE

- `class SubClass(BaseClass):`
- "is-a" relationship - a subclass is a type of base class
- By default, a subclass has the same behavior as its base class
- To make a subclass different from its base class:
 - Add attributes
 - declare additional methods/variables within the subclass
 - Override attributes
 - Class variables - reassign
 - Methods - redefine the method with the same function signature (name and arguments)

CALLING METHODS FROM THE BASE CLASS

When defining a method, we may want to reuse the method from the base class first, then add more to it

- `super().method(args)`
 - Can only be used inside of a class method
 - no need to pass it in `self`
- `BaseClass.method(instance, args)`
 - Can be used anywhere
 - Need to explicitly pass in the `instance`

```
class Dog(Pet):  
    def __init__(self, name, owner, has_floppy_ears):  
        super().__init__(name, owner)  
        # alternatively, Pet.__init__(self, name, owner)  
        self.has_floppy_ears = has_floppy_ears
```

NOW IT'S LAB TIME 🤠

- Get started on the lab and raise your hand whenever you need help!
- Get to know your neighbors and collaborate if you'd like!
- Slides: go.cs61a.org/mingxiao-index
- Leave any anonymous feedback here: go.cs61a.org/mingxiao-anon

AND REMEMBER TO GET
CHECKED OFF! 🧺

go.cs61a.org/mingxiao-att

The secret phrase is ...
(NOT 3 dots! I'll announce it 🙊)