Lecture 24

CLASSES PT. 2

OBJECT-ORIENTED PROGRAMMING

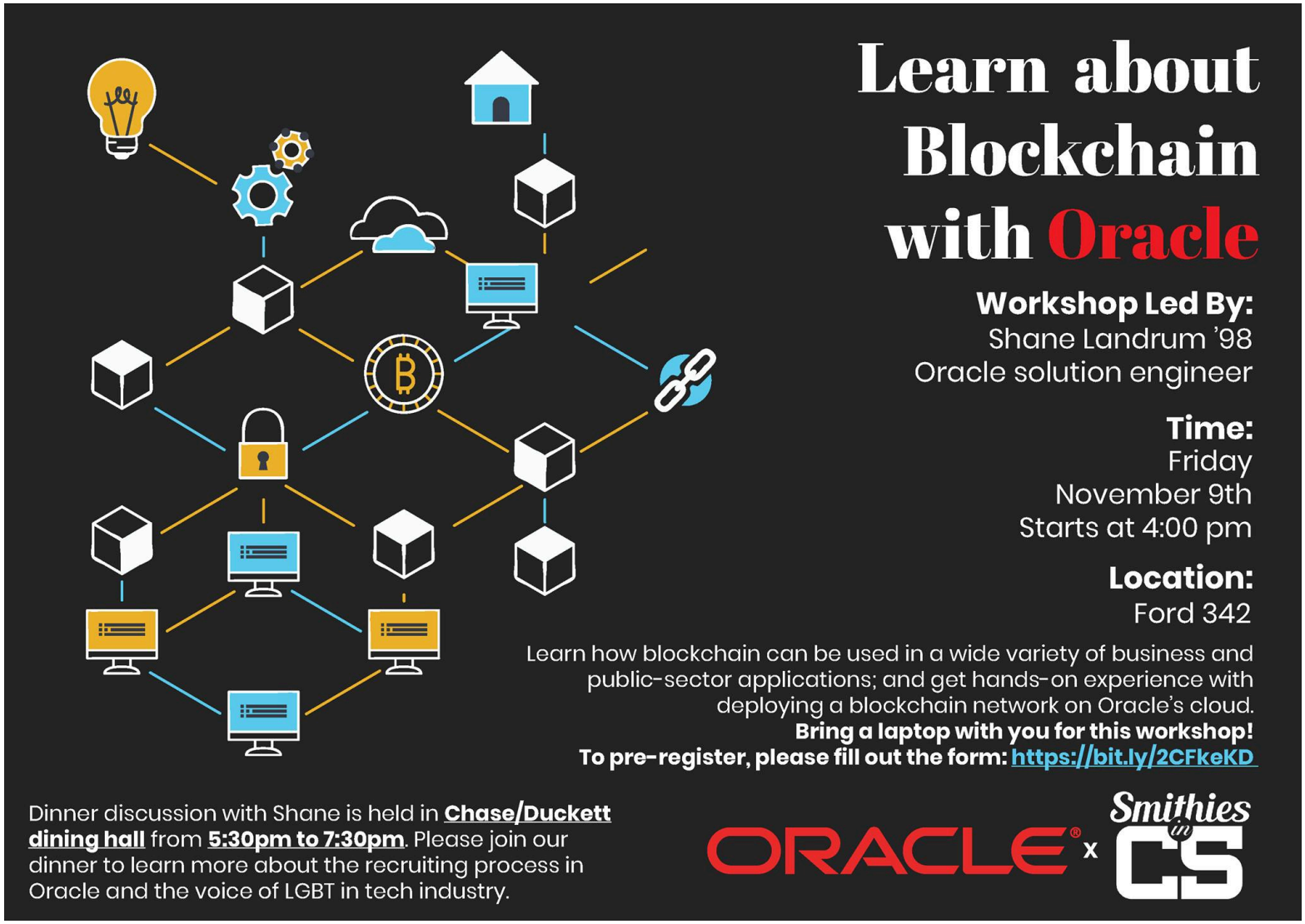
CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

Reminder


Learn about Blockchain with Oracle
Workshop Led By:
Shane Landrum '98
Oracle solution engineer
Time:
Friday
November 9th
Starts at 4:00 pm
Location:
Ford 342
Learn how blockchain can be used in a wide variety of business and public-sector applications; and get hands-on experience with deploying a blockchain network on Oracle's cloud.
Bring a laptop with you for this workshop!
To pre-register, please fill out the form: https://bit.ly/2CFkeKD
Dinner discussion with Shane is held in Chase/Duckett dining hall from 5:30pm to 7:30pm. Please join our dinner to learn more about the recruiting process in Oracle and the voice of LGBT in tech industry.
ORACLE x
Smithies in CS

Questions about the homework

Q: “I don’t know anything about Pokémon, so I’m confused about the different variables… can you explain them?”

A: Yes!

Questions about the homework

0 `max_hp`

hp (“hit points”)

= how **strong** your Pokémon is

(bigger number → **stronger**)

Questions about the homework

0 `max_hp`

`attack_power`

= how **much damage** your Pokémon does to another when attacking
(bigger number → causes **more damage**)

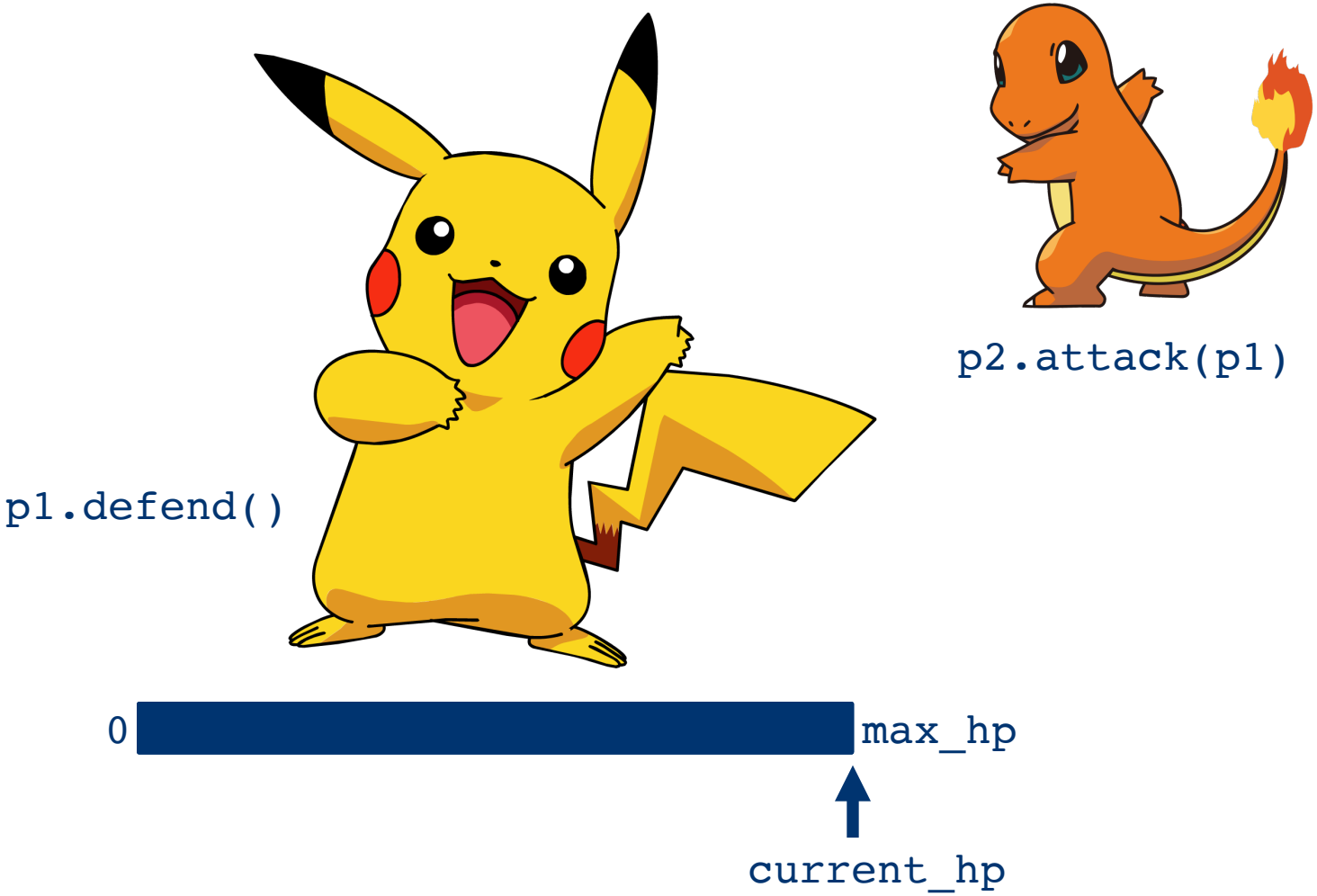
Questions about the homework

0  `max_hp`

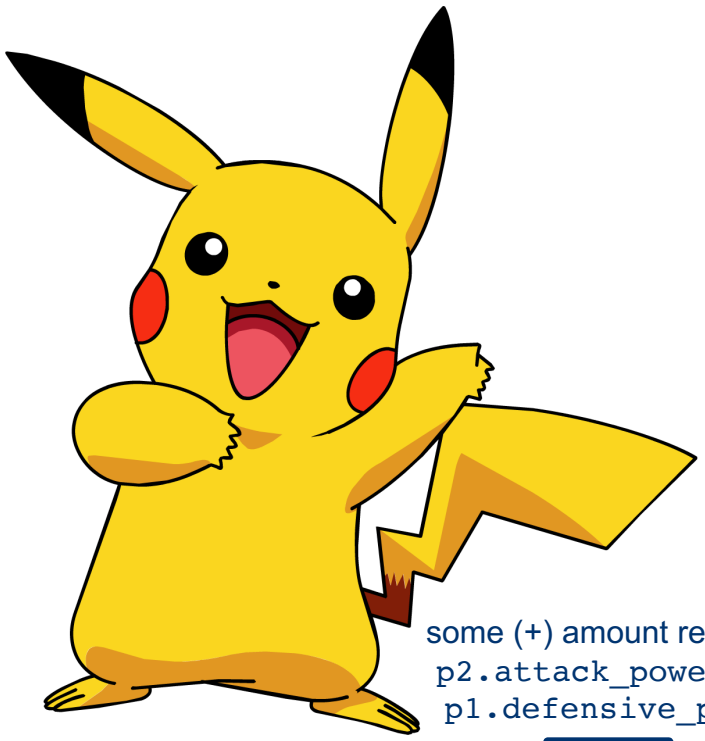
`defensive_power`

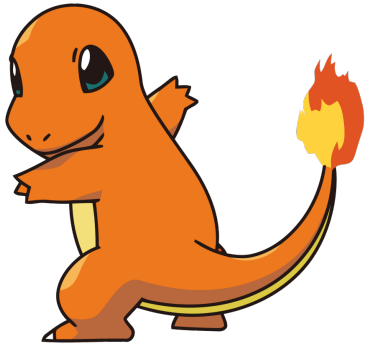
= how **easy it is for** your Pokémon to fend off damage from another's attack
(bigger number → takes **less damage**)

Questions about the homework



Questions about the homework

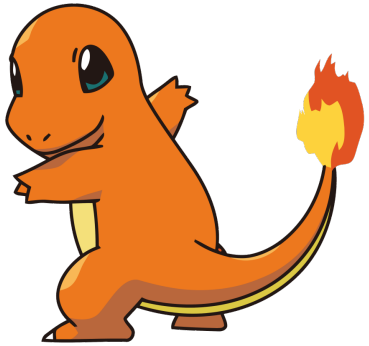




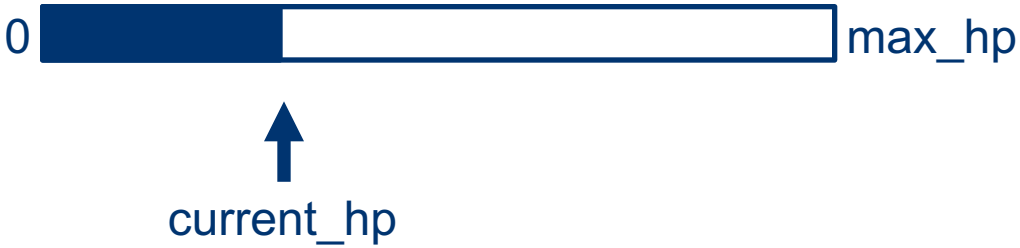
`p2.attack(p1)`

some (+) amount related to `p2.attack_power` **and** `p1.defensive_power`

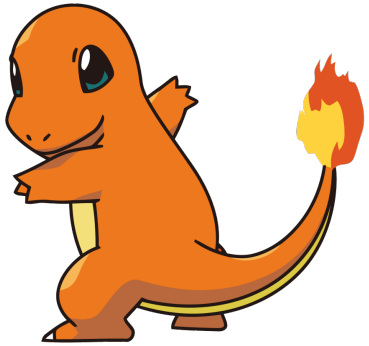
Questions about the homework



`p2.attack(p1)`

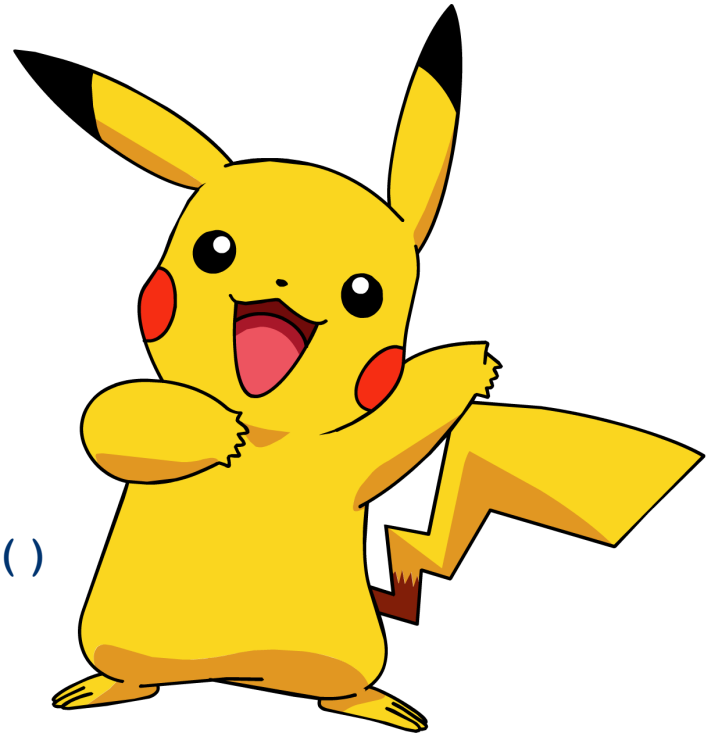


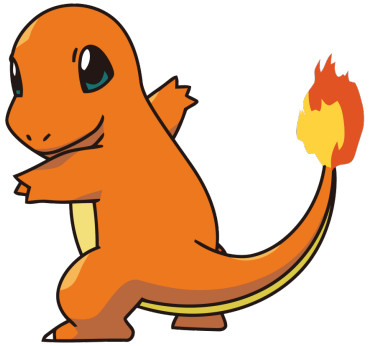
Questions about the homework



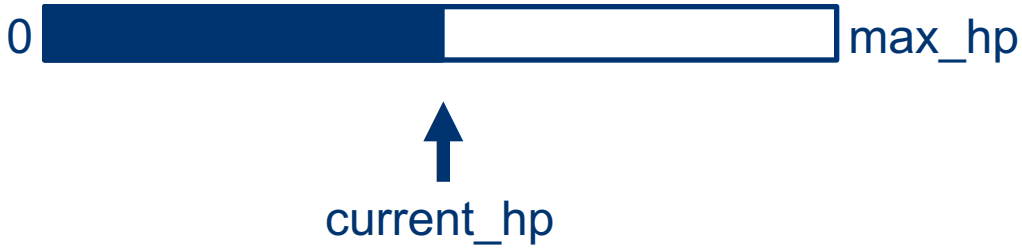
fainted = `True`

Questions about the homework





`p1.revive()`



Universal extension on A7

- These are big, complicated ideas
- To give you more time to wrestle with them, I'm granting an extension until **Tuesday** at 11:55pm
  - This means you have two extra sessions of TA hours
  - And one more session of office hours with me
- **HOWEVER**, this doesn't mean that we're going to slow everything else down… so budget accordingly!

Discussion

Still have questions?

Outline

- ✓HW 6 recap
- ✓Classes pt. 1: attributes and methods
- ✓Classes pt. 2: object-oriented programming
  - ✓big idea
  - ✓recap: `classes`
  - ✓public vs. private
- Lab: Classy Playlist
- Classes pt. 3: inheritance
  - child classes
  - overriding parent attributes / methods

Lab 8: Classy Playlist

How did it go?

RECAP: `class` definitions (“blueprints”)

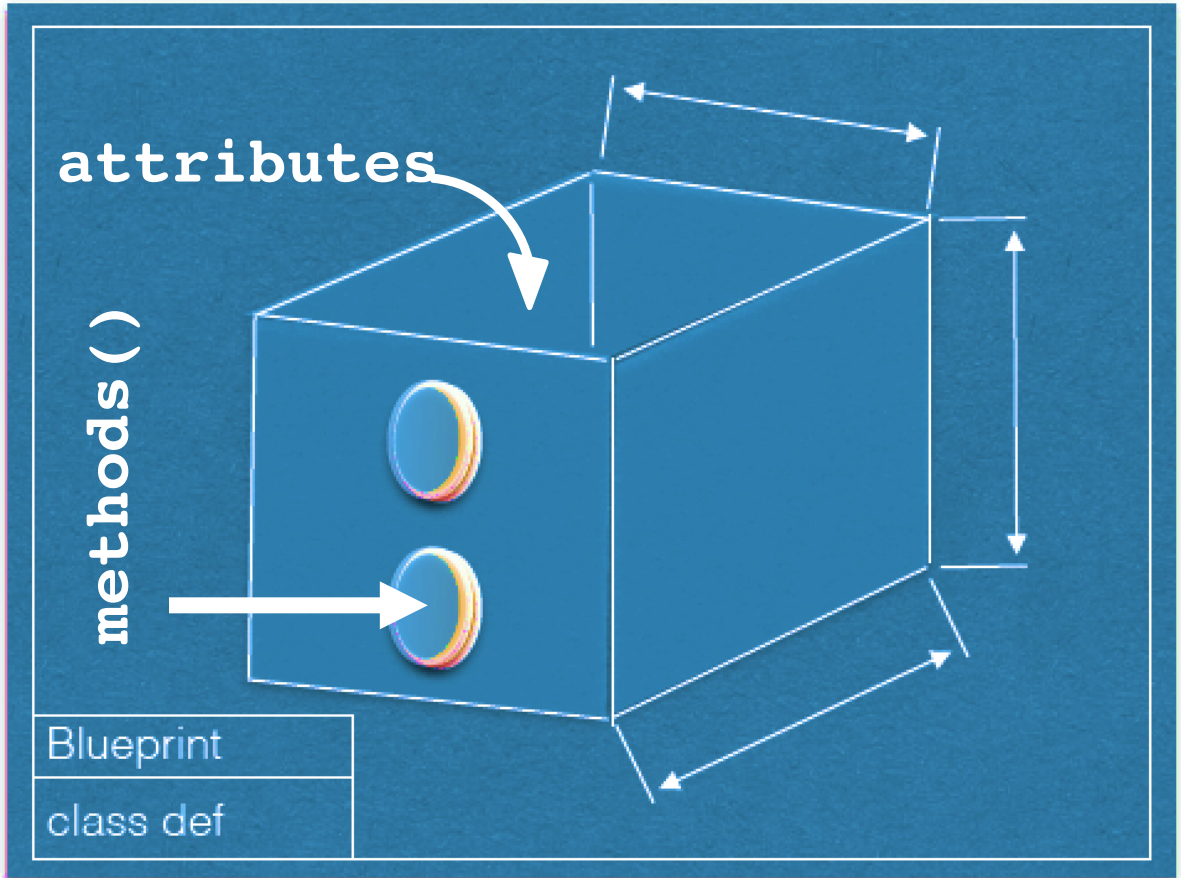


Image courtesy Dominique Thiebaut, Smith College

Coding the **Song class**

*classyPlaylist.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan-CSC/labs/classy...

```
class Song:

    def __init__(self, title, artist, url, duration):
        self.title = title
        self.artist = artist
        self.url = url
        self.duration = duration

    def print(self):
        print("'" + self.title + "'", end = " ")
        print("by " + self.artist, end = " ")
        print("(" + self.duration + ")")

    def play(self):
        print("Now playing", end = " ")
        self.print()
        webbrowser.open(self.url)
        sleep(self.total_seconds)
        print("Song is over!")
```

Ln: 19 Col: 32

Coding the **Song class**

the constructor

*classyPlaylist.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan-CSC/labs/classy...

```
class Song:

    def __init__(self, title, artist, url, duration):
        self.title = title
        self.artist = artist
        self.url = url
        self.duration = duration

    def print(self):
        print("'" + self.title + "'", end = " ")
        print("by " + self.artist, end = " ")
        print("(" + self.duration + ")")

    def play(self):
        print("Now playing", end = " ")
        self.print()
        webbrowser.open(self.url)
        sleep(self.total_seconds)
        print("Song is over!")
```

Ln: 19 Col: 32

Coding the **Song class**

*classyPlaylist.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan-CSC/labs/classy...

```
class Song:

    def __init__(self, title, artist, url, duration):
        self.title = title
        self.artist = artist
        self.url = url
        self.duration = duration

    def print(self):
        print("'" + self.title + "'", end = " ")
        print("by " + self.artist, end = " ")
        print("(" + self.duration + ")")

    def play(self):
        print("Now playing", end = " ")
        self.print()
        webbrowser.open(self.url)
        sleep(self.total_seconds)
        print("Song is over!")
```

Ln: 19 Col: 32

attributes

Coding the **Song class**

*classyPlaylist.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan-CSC/labs/classy...

```
class Song:

    def __init__(self, title, artist, url, duration):
        self.title = title
        self.artist = artist
        self.url = url
        self.duration = duration

    def print(self):
        print("'" + self.title + "'", end = " ")
        print("by " + self.artist, end = " ")
        print("(" + self.duration + ")")

    def play(self):
        print("Now playing", end = " ")
        self.print()
        webbrowser.open(self.url)
        sleep(self.total_seconds)
        print("Song is over!")
```

Ln: 19 Col: 32

methods

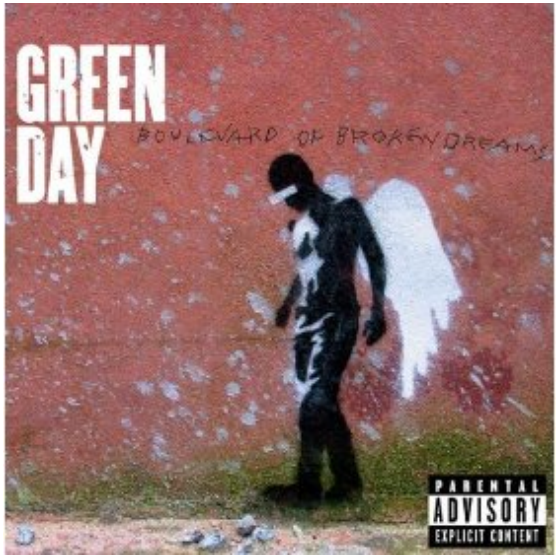
Creating a **`Song instance`**

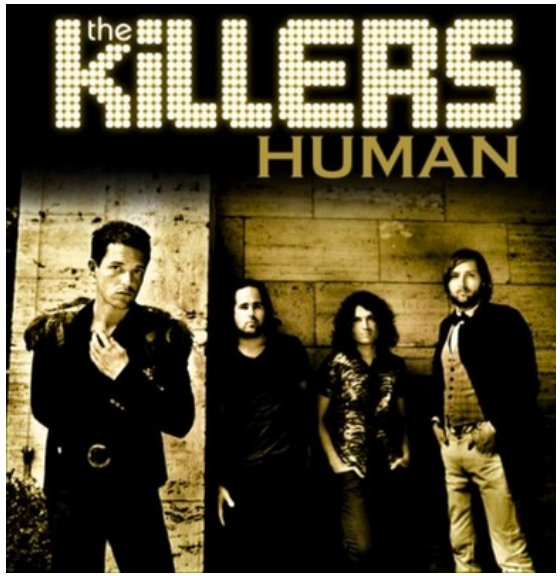
*classyPlaylist.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

```
def main():
    # Get user input
    title = input("Title? ")
    artist = input("Artist? ")
    url = input("Link to YouTube video? ")
    duration = input("Duration? ")

    # Create Song and return it
    return Song(title, artist, url, duration)

main()
```

Ln: 46 Col: 0

Lots of possible `Song instances`





All from the **same blueprint**

*classyPlaylist.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan-CSC/labs/classy...

```
class Song:

    def __init__(self, title, artist, url, duration):
        self.title = title
        self.artist = artist
        self.url = url
        self.duration = duration

    def print(self):
        print("'" + self.title + "'", end = " ")
        print("by " + self.artist, end = " ")
        print("(" + self.duration + ")")

    def play(self):
        print("Now playing", end = " ")
        self.print()
        webbrowser.open(self.url)
        sleep(self.total_seconds)
        print("Song is over!")
```

Ln: 19 Col: 32

Outline

- ✓HW 6 recap
- ✓Classes pt. 1: attributes and methods
- ✓Classes pt. 2: object-oriented programming
  - ✓big idea
  - ✓recap: `classes`
  - ✓public vs. private
- ✓Lab: Classy Playlist
- Classes pt. 3: inheritance
  - child classes
  - overriding parent attributes / methods

Motivation

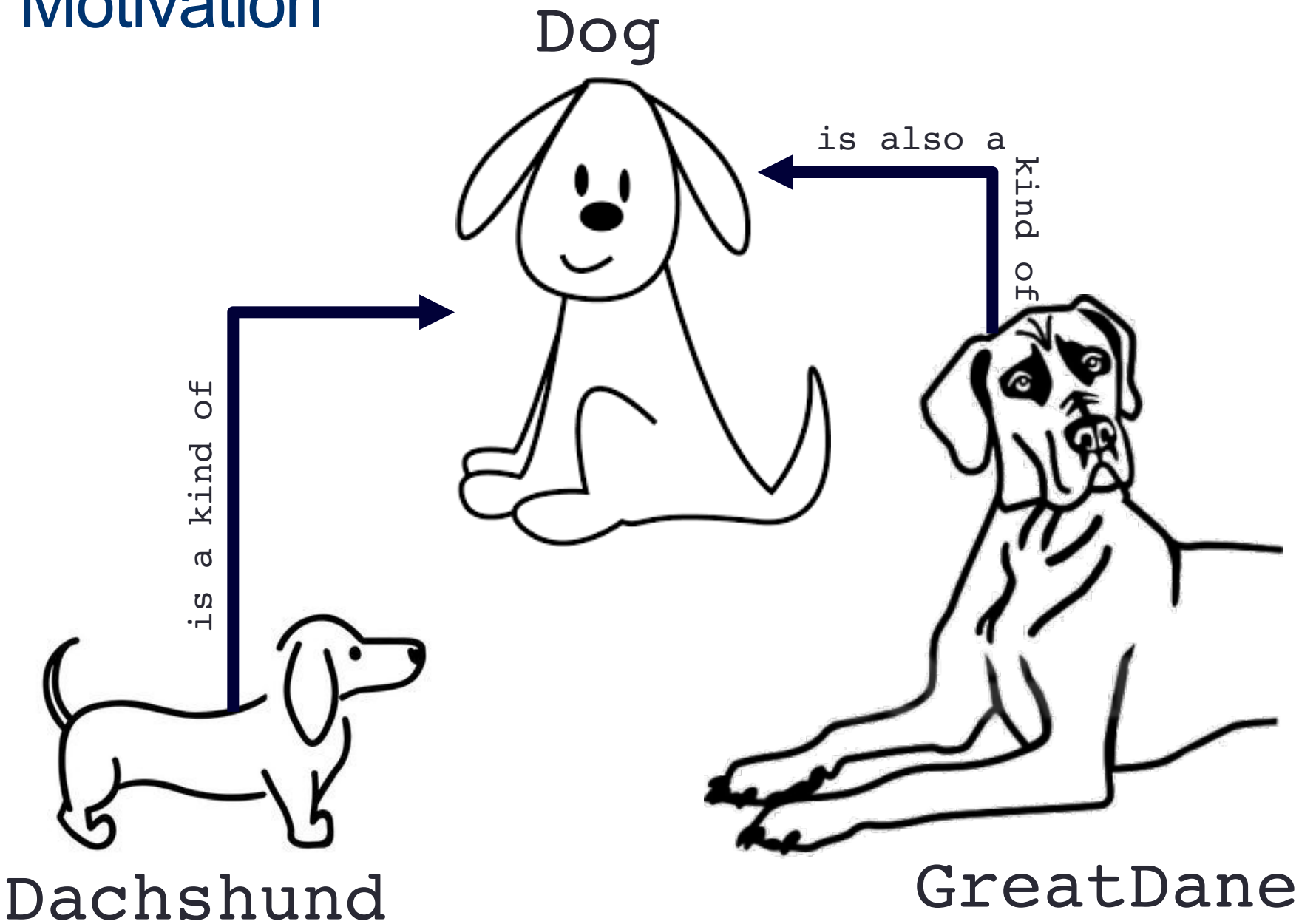
10 minute exercise: the `Dog` class

- Write a class called **`Dog`**, with a constructor that takes in the following parameters:

name (the dog's name)

age (the dog's age in years)

Motivation



As (sub)**classes**

Untitled

```
class Dog:

    # A class attribute (every Dog has the same value)
    species = "Canine"

    def __init(self, name, age):
        self.name = name
        self.age = age


class Dachshund(Dog):
    def run():
        print("I'm running low to the ground!")


class GreatDane(Dog):
    def leapOver(something):
        print("I'm leaping over", something)
```

Ln: 15 Col: 0

As (sub)**classes**

```
class Dog:

    # A class attribute (every Dog has the same value)
    species = "Canine"

    def __init(self, name, age):
        self.name = name
        self.age = age


class Dachshund(Dog):
    def run():
        print("I'm running low to the ground!")


class GreatDane(Dog):
    def leapOver(something):
        print("I'm leaping over", something)
```

subclasses "**inherit**" all the **attributes** and **methods** from their parent **class**

As (sub)**classes**

```
class Dog:

    # A class attribute (every Dog has the same value)
    species = "Canine"

    def __init(self, name, age):
        self.name = name
        self.age = age


class Dachshund(Dog):
    def run():
        print("I'm running low to the ground!")


class GreatDane(Dog):
    def leapOver(something):
        print("I'm leaping over", something)
```

they can also have **their own** **`attributes`** and **`methods`** separate from their parent

As (sub)**classes**

Untitled

```
class Dog:

    # A class attribute (every Dog has the same value)
    species = "Canine"

    def __init(self, name, age):
        self.name = name
        self.age = age

class RobotDog(Dog):
    species = "Robot'
```

Ln: 9 Col: 0

if necessary, they can override **attributes** and **methods** from their parent

Discussion

Why is this ”inheritance” idea **useful**?

Coming up next

- Monday: python packages (graphics)
- Wednesday: animation
- Lab: Fish Tank
- Friday: Interaction