Lecture 23

CLASSES PT. 3

INHERITANCE

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

Outline

- ✓Friday: Classes pt. 1: attributes and methods
- ✓Today: Classes pt. 2: object-oriented programming
  - ✓big idea
  - ✓recap: `classes`
  - ✓public vs. private
- Lab: working with classes
- Wednesday: Classes pt. 3: inheritance
  - child classes
  - overriding parent attributes / methods
- Friday: Introduction to Algorithms

Lab 7: working with classes

How did it go?

RECAP: `class` definitions ("blueprints")

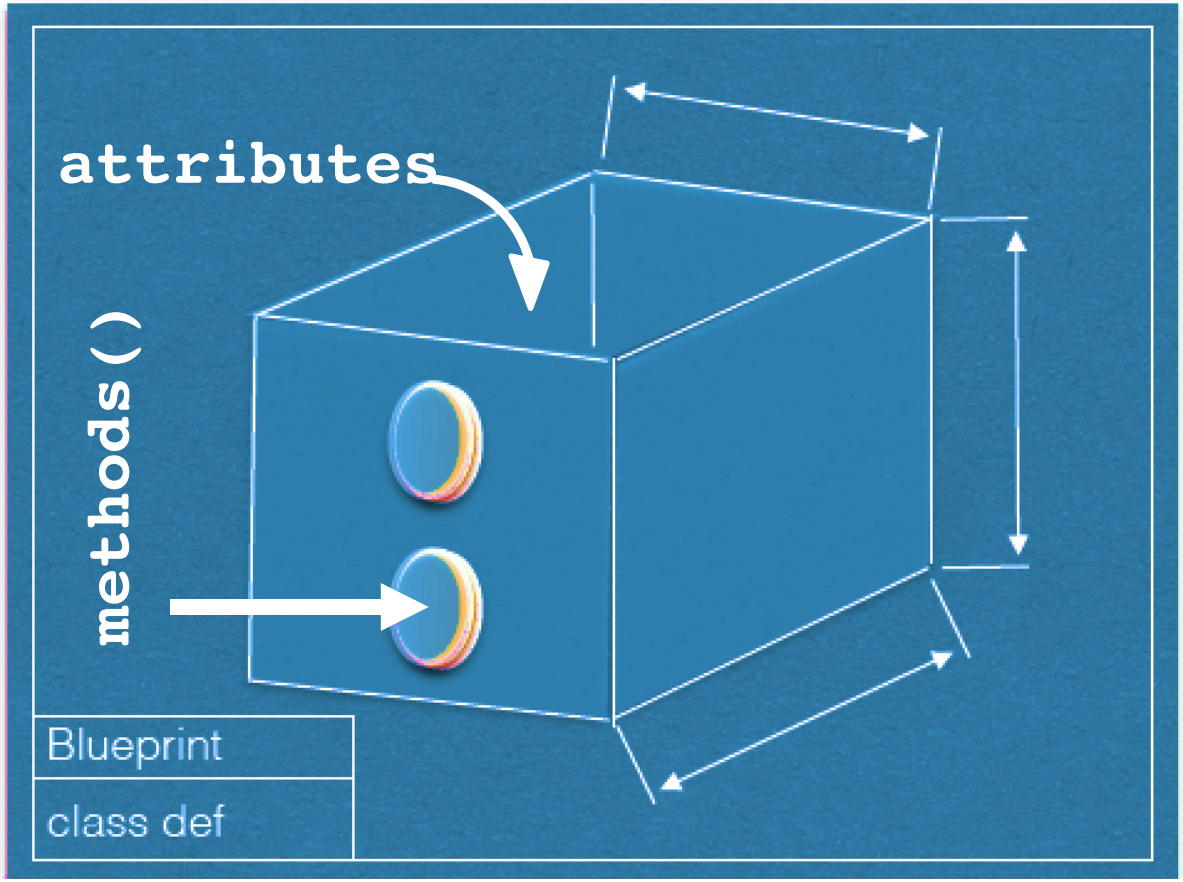


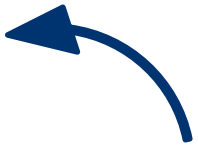
Image courtesy Dominique Thiebaut, Smith College

Coding the `Artist` class

```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

Coding the `Artist` class

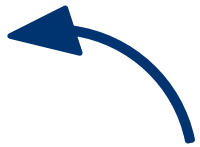
```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```



the constructor

Coding the `Artist` class

```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```



attributes

Coding the `Artist` class


```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

Coding the `Artist` class

```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year ==
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

method

Creating an `Artist` instance

```
if __name__ == "__main__":
    user_artist_name = input()
    user_birth_year = int(input())
    user_death_year = int(input())
    user_title = input()
    user_year_created = int(input())

    user_artist = Artist(user_artist_name, user_birth_year, user_death_year)
```

Lots of possible **`Artists`**

All from the **same blueprint**

```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

Outline

- ✓Friday: Classes pt. 1: attributes and methods
- ✓Today: Classes pt. 2: object-oriented programming
  - ✓big idea
  - ✓recap: `classes`
  - ✓public vs. private
- ✓Lab: working with classes
- Wednesday: Classes pt. 3: inheritance
  - child classes
  - overriding parent attributes / methods
- Friday: Introduction to Algorithms

Motivation

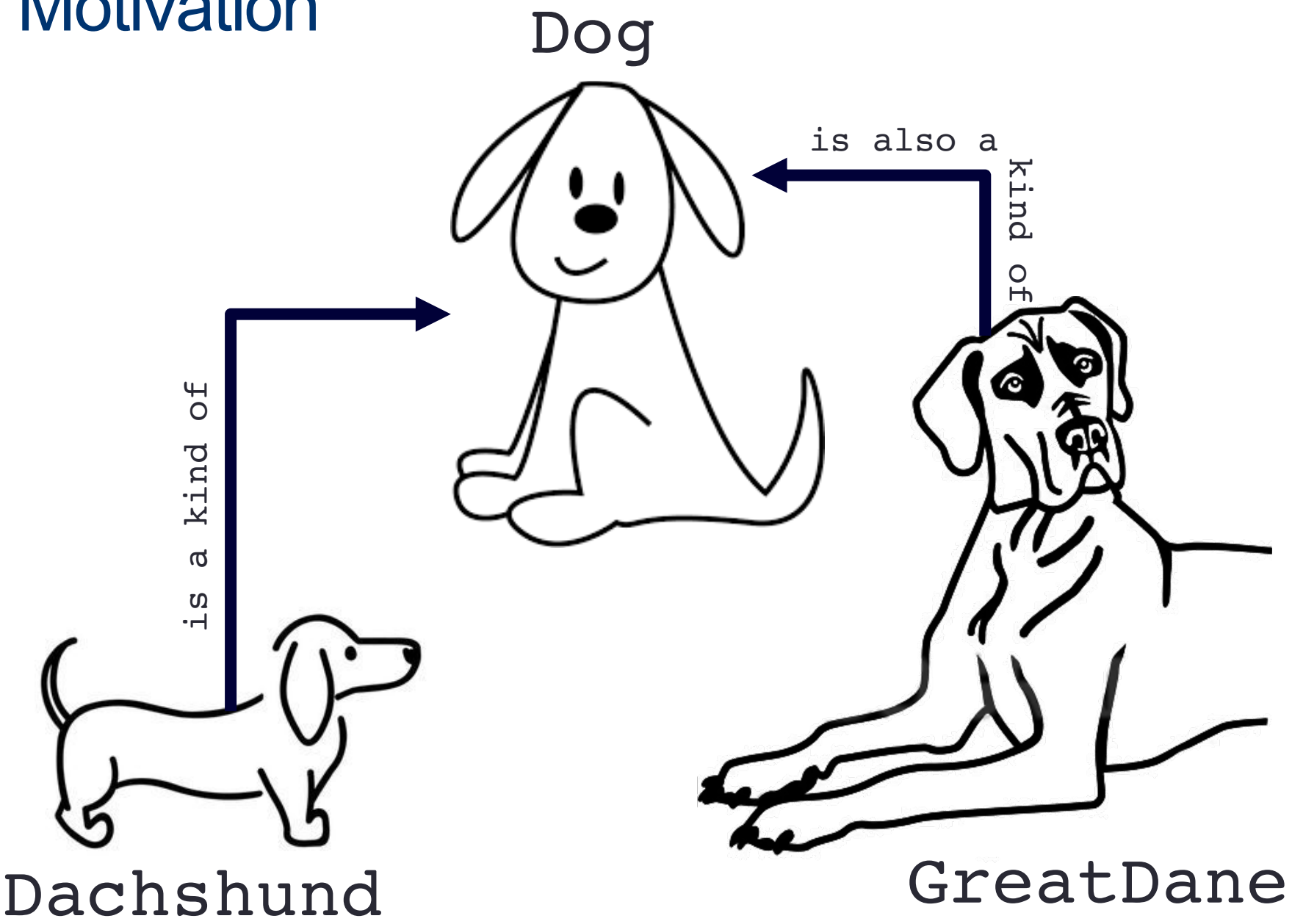
10 minute exercise: the `Dog` class

- 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

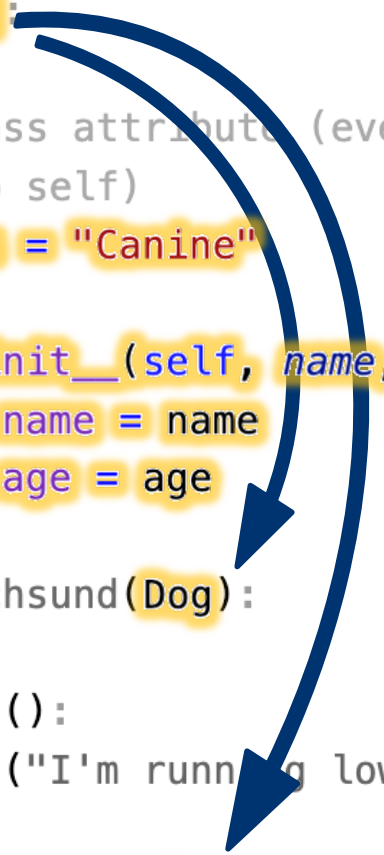


As (sub)**classes**

```
class Dog:

  # A class attribute (every Dog has the same value,
  # so no self)
  species = "Canine"

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

class Dachsund(Dog):

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

class GreatDane(Dog):

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

As (sub)**classes**

```
class Dog:

  # A class attribute (every Dog has the same value,
  # so no self)
  species = "Canine"

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

class Dachsund(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,
  # so no self)
  species = "Canine"

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

class Dachsund(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**

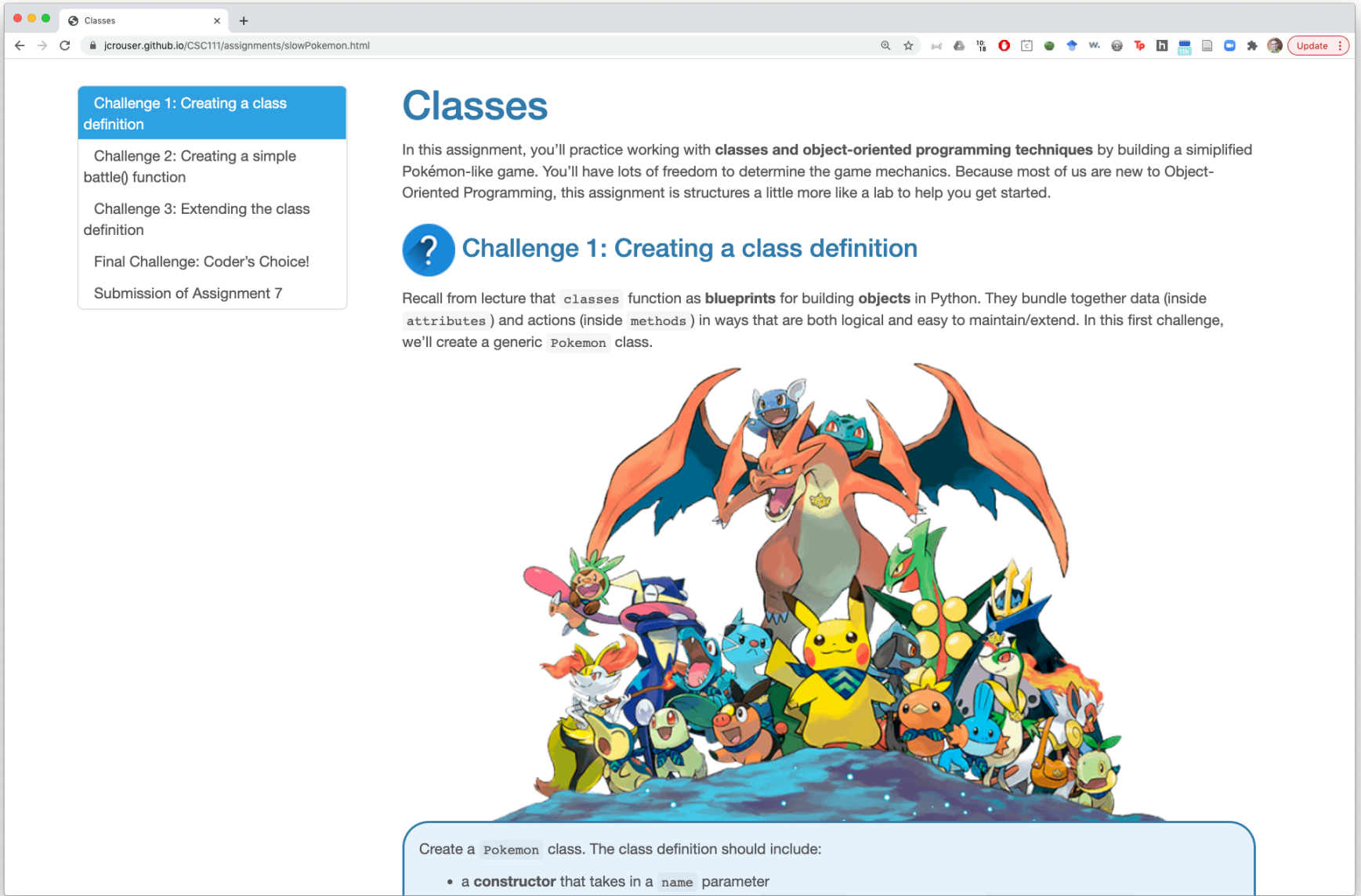
```
class Dog:

  # A class attribute (every Dog has the same value,
  # so no self)
  species = "Canine"

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

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

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

Discussion

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

A6: Slow Pokémon (open a little early)



Common questions about the homework

0  `max_hp`

hp (“hit points”)

= how **strong** your Pokémon is

(bigger number → **stronger**)

Common questions about the homework

0  `max_hp`

`attack_power`

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

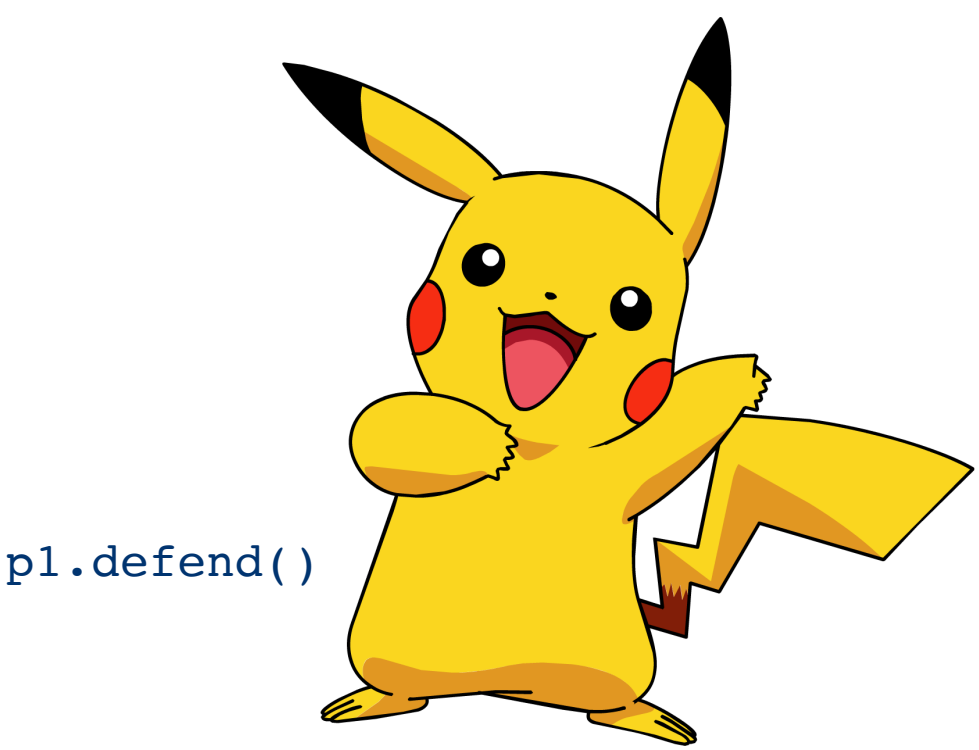
Common questions about the homework

0  `max_hp`

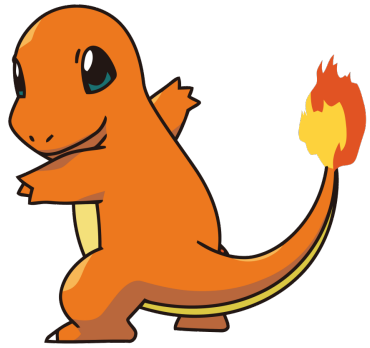
`defensive_power`

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

Common questions about the homework

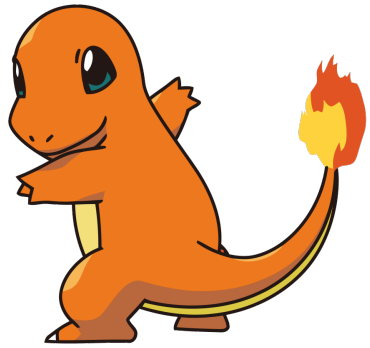


`p1.defend()`



`p2.attack(p1)`

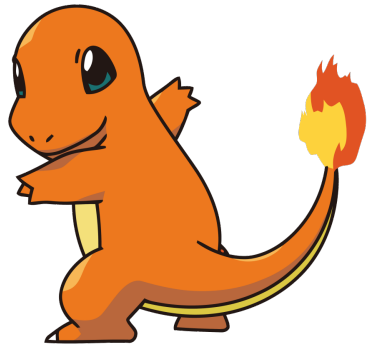
Common questions about the homework



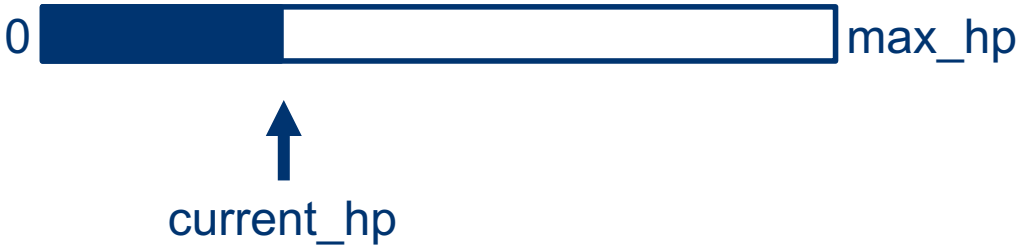
`p2.attack(p1)`

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

Common questions about the homework



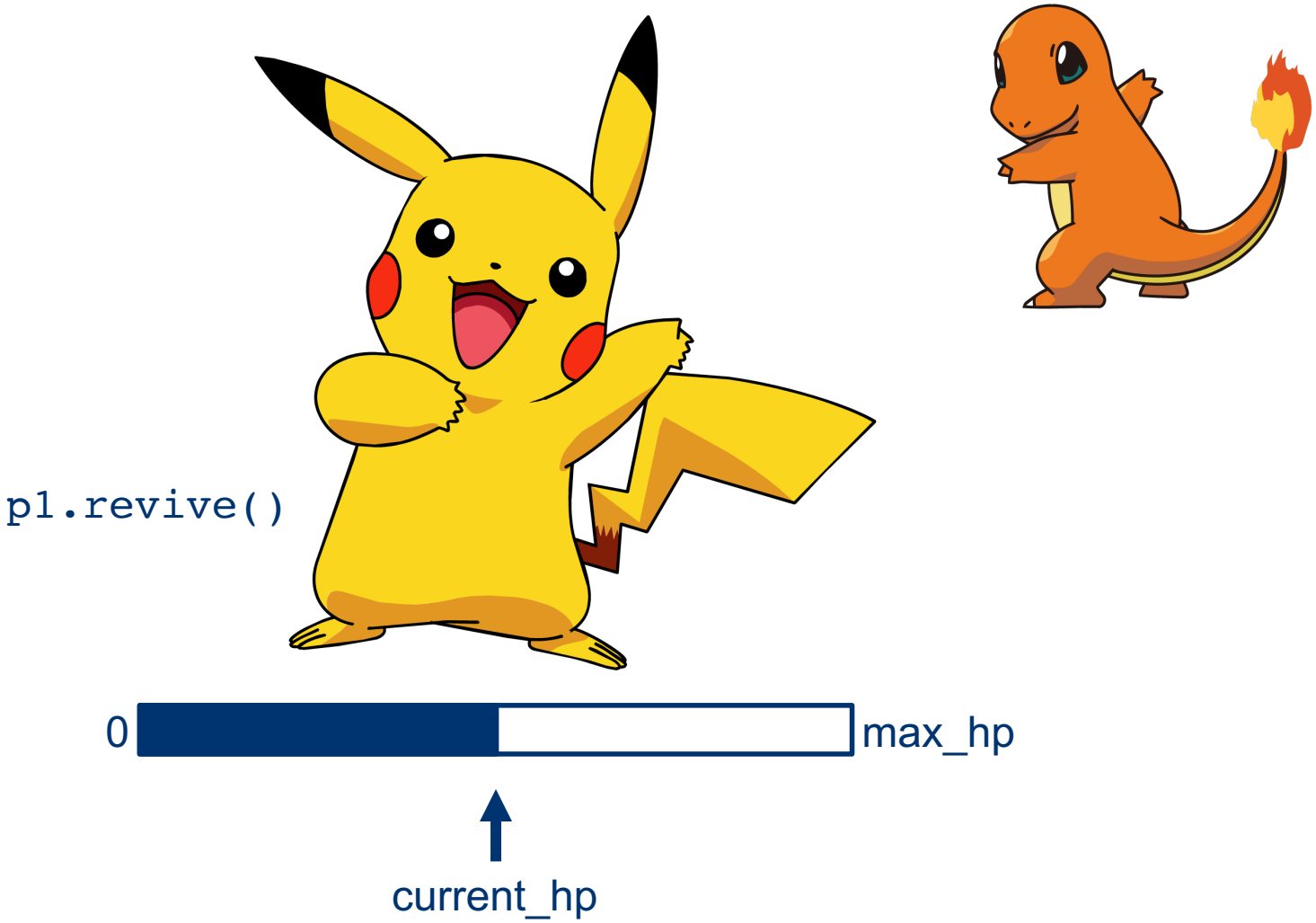
```
p2.attack(p1)
```



Common questions about the homework

fainted = `True`

Common questions about the homework



Outline

- ✓Friday: Classes pt. 1: attributes and methods
- ✓Today: Classes pt. 2: object-oriented programming
  - ✓big idea
  - ✓recap: `classes`
  - ✓public vs. private
- ✓Lab: working with classes
- ✓Wednesday: Classes pt. 3: inheritance
  - ✓child classes
  - ✓overriding parent attributes / methods
- Friday: Introduction to Algorithms