Week 12 - Monday

COMP 1800

Last time

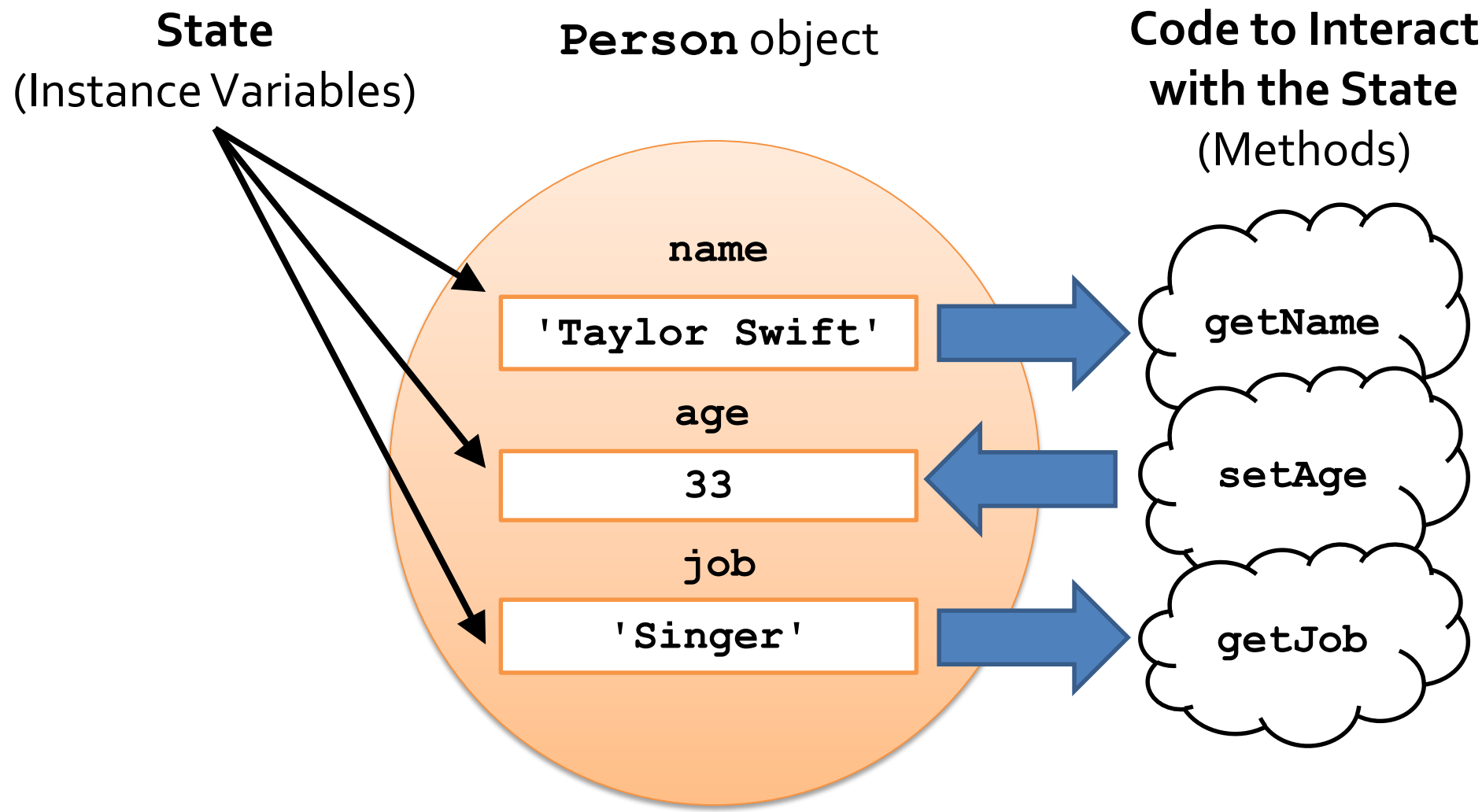
- What did we talk about last time?
- Exam 2 post mortem
- Recursion practice
  - Sierpinski triangle!
- Work time

Questions?

Assignment 8

Objects in Python

What's an object?



Objects

- The idea of an object is to group together data and code
- You have used objects a bit already
  - Strings are objects
  - Even lists are a special kind of object

Why are objects a good idea?

- Encapsulation: hiding data to keep it safe
- Methods provide useful ways to interact with the data
- It's convenient to keep related data grouped together
  - You could have a list of **`Person`** objects instead of three separate lists of names, ages, and jobs

Calling methods

- When you have an object, you can call methods on it
- A method is like a function, except that it has access to the details of the object
- To call a method, you type the name of the object, a dot, and the name of the method
- A method will always have parentheses after it
- Sometimes the parentheses will have arguments that the method uses

Method call examples

- You've called methods with strings:

```
phrase = 'BOOM goes the dynamite!'
other1 = phrase.lower() # gets lowercase version
other2 = phrase.upper() # gets uppercase version
words = phrase.split() # turns to list
```

- You've called methods on a list:

```
words.sort() # sorts the list
```

Instance variables

- Instance variables are the data **inside** of an object
- Like methods, you can access an instance variable with the name of the object, a dot, and then the name of the member
- Unlike methods, instance variables never have parentheses
- They are values, not functions that do things

Adding members

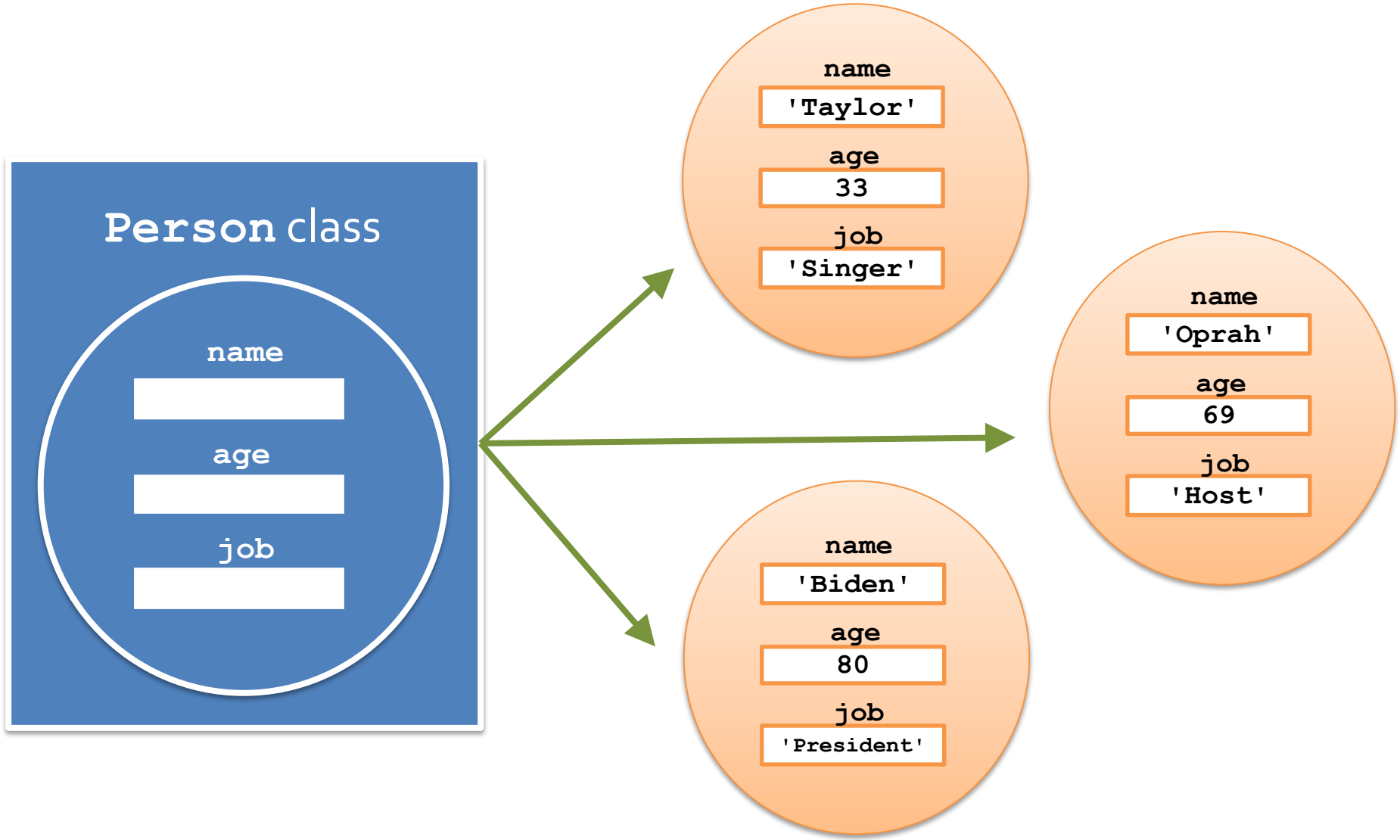
- Python allows us to add instance variables anytime we want
- Doing so lets us keep extra information in each object
- For example, we could give a **`Person`** object a **`nickname`** variable after creating it

```
taylor = Person('Taylor Swift', 33, 'Singer')
taylor.nickname = 'Tay Tay'
```

Creating entirely new classes

- Adding instance variables is fine, but what if you want to create an object from scratch?
- A **class** is a template for an object
- You can define a class that will allow you to create your own custom objects

Classes are like blueprints



Planet class

- Let's look at an example class that holds information about a planet

```
class Planet:
    def __init__(self, name, radius, mass, distance):
        self.name = name
        self.radius = radius
        self.mass = mass
        self.distance = distance

    def getName(self):
        return self.name

    def setName(self, name):
        self.name = name
```

What is `self`?

- **`self`** is a reference to the object that you're currently inside of
- If you forget to use **`self`**, you aren't talking about the current object, you're talking about an outside variable
- The Java or C++ equivalent of **`self`** is **`this`**
- When calling a method (or the constructor), you always ignore the **`self`** parameter
- The object itself is automatically supplied

Constructor

- A **constructor** is a special kind of method that initializes the values inside of an object
- It's how a new object is created
- In Python, its name is always **`__init__`**
- It takes in the initial values for the object

```
class Planet:
    def __init__(self, name, radius, mass, distance):
        self.name = name
        self.radius = radius
        self.mass = mass
        self.distance = distance
```

Creating a new object

- To create a new object, you call its constructor
- This means typing the name of the class with parentheses after it, including the initial values for the object
- When you call the constructor, you **don't** pass in **`self`**!
  - That happens automatically

```
planet1 = Planet('Jupiter', 69911, 1.9E27, 7.78E8)
planet2 = Planet('Mars', 3390, 6.4e23, 2.27E8)
```

Accessors

- An **accessor** is a kind of method that **gets** a value out of an object
- It can read an existing value or compute a new one
- An accessor doesn't change the data inside the object

```
def getName(self):
    return self.name
```

- Calling an accessor is like calling any other method on an object
  - Object name, dot, then method name
  - Leave off the **self**!

```
name = planet1.getName()
print(name)
```

Mutators

- A **mutator** is a kind of method that **sets** a value in an object
- Its purpose is to change the data inside the object

```
def setName(self, name):
    self.name = name
```

- It could do some checking to make sure that a good value is supplied

```
planet1.setName('Jove')   # new name
print(planet1.getName()) # prints Jove
```

Let's write some accessors

- We need accessors for:
  - Radius
  - Mass
  - Distance

Let's write more!

- Accessors don't have to report instance variables as they are
- They could also combine instance variables to answer questions
- Using formulas, we can find
  - Volume: $\frac{4}{3}\pi r^3$
  - Surface area: $4\pi r^2$
  - Density: $\frac{m}{V}$

Special methods

- Python uses a number of special methods
  - A constructor (`__init__`) is one
- What happens if you try to print a `Planet` object?
  - `<__main__.Planet object at 0x0000000030D4080>`
  - Not very helpful
- There's a special `__str__` method that gives back a string version of the object
- Let's make one that gives back the name

Planets

- We can make a number of planets using the following data

Name	Radius (km)	Mass (kg)	Distance (km)
Mercury	2440	3.3E23	5.79E7
Venus	6052	4.9E24	1.08E8
Earth	6371	6.0E24	1.50E8
Mars	3390	6.4E23	2.28E8
Jupiter	69911	1.9E27	7.78E8
Saturn	58232	5.7E26	1.42E9
Uranus	25362	8.7E25	2.87E9
Neptune	24622	1.0E26	4.50E9

Lists

- It's convenient to put objects in lists
- We could have a list containing all the planets we made:

```
planets = [mercury, venus, earth, mars, jupiter,
saturn, uranus, neptune]
```

Determine biggest planet

- With all the planets in a list, we could do something useful, like find the biggest planet
  - Obviously, this might be more interesting if the list were bigger

```
biggest = planets[0]
for planet in planets:
    if planet.getRadius() > biggest.getRadius():
        biggest = planet

print('Biggest:", biggest)
```

`Student` class

- Let's write a **`Student`** class
- Instance variables:
  - First Name
  - Last Name
  - GPA
  - ID
- We need accessors for all of the instance variables
- And mutators for GPA

More special methods

- Python has other special methods
- Some are useful if your class is designed to hold a collection of things
  - The **`__getitem__`** method retrieves an item based on the index specified
  - The **`__len__`** method returns the number of items in the collection
  - The **`__contains__`** method says whether or not an element is in your collection

Sentence class

- Let's make a **`Sentence`** class
- Its constructor
  - Takes a string
  - Splits that string on spaces to make a list of strings
  - Stores that list as its instance variable
- The **`__getitem__`** method should return the specified words in the list
- The **`__len__`** method returns the number of words in the sentence
- The **`__contains__`** method should say whether the list contains the string the user is looking for

Upcoming

Next time…

- Animating the solar system

Reminders

- **Vote!**
- Read sections 10.4, 10.5, and 10.6
- Keep working on Assignment 8