Inheritance and Overriding (and some virtual env and debugging stuff)

What we’re going to learn

- What inheritance is
- How to implement and use inheritance
- When to use inheritance
  - Inheritance vs. composition
- What overriding is and how to implement it
- What venv is in Python and why it’s used
- How to use VS Code’s debugger

Check In

Write your favorite movie or video game soundtrack that you like to study to

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Recap of OOP: Fill in the blanks

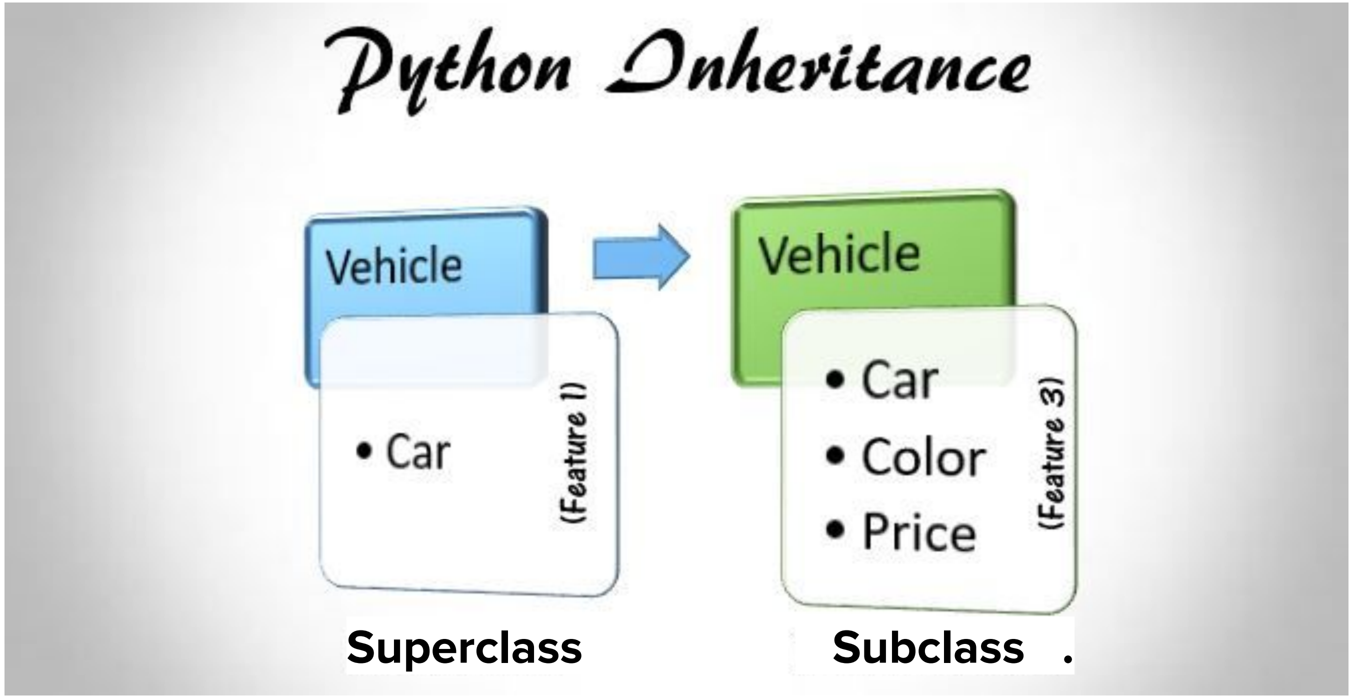
- An **object** is __
- **Classes** are __
- Classes can be utilized by ____ them into computer memory.

Recap of OOP Part 1: Encapsulation

- An **object** is a way to organize a collection of variables (called **properties**) and functions (called **methods**) that can act on the object's properties. This concept is known as **encapsulation**, which helps manage complexity.
- **Classes** are blueprints (definitions) of all objects of the same type.
- Classes can be utilized by **instantiating** them into computer memory.
- **Instantiation** of a class creates it in memory and allows users to explicitly manipulate the object.

Object Inheritance

- **Inheritance** is a mechanism of objects that allows one to derive classes from other classes and share methods and properties among different class hierarchies.
- An important distinction with object inheritance involves identifying **subclasses** and **superclasses**.
  - A **superclass** is a higher-hierarchy base class that has basic attributes to derive from.
  - A **subclass** is a lower-hierarchy class that derives attributes from another base class.



Class inheritance for the **`Vehicle`** class.

Pac-Man with OOP

Consider the famous arcade game **Pac-Man**.

Pac-Man can be used to illustrate class hierarchies and conventions.

Let's consider two classes: **`PacMan`** and **`Pellet`**.

Pac-Man must eat **`Pellet`** instances.

`Pellet` instances have a property called **`.is_eaten`**.

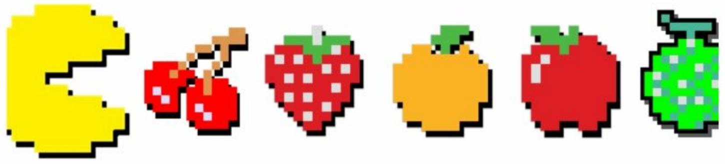
Pac-Man and Inheritance

Consider our **Pac-Man** example.

Let’s extend the classes to now include some examples of inheritance.

The **`Pellet`** class can be a subclass of a more generalized **`Eatable`** class.

This allows us to create some other eatable objects, including a **`Fruit`** object and a **`Ghost`** class.

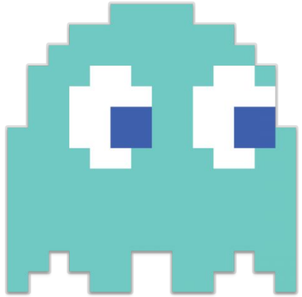


Pac-Man and Inheritance

Since the **`Eatable`** class has a property called **`.is_eaten`**, its subclasses *inherit* that property.

The **`Fruit`** class can have its own properties and methods (like **`.move()`**), and also *inherits* the **`.is_eaten`** property from its **`Eatable`** superclass.

The **`Ghost`** class can have its own properties (like **`.speed`**, **`.is_alive`**, **`.is_scared`**) and methods (like **`.move()`**), and also *inherits* the **`.is_eaten`** property from **`Eatable`**.

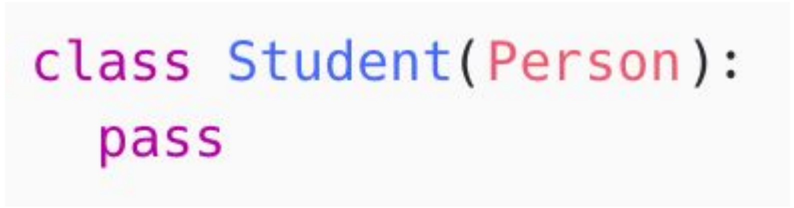


Activity: Model a Team with OOP

1. Form a group of **3-4 students** and write your response in pear deck.
2. Select a **team sport (or organization)** that involves people in **several different roles**. (Ex: soccer, field hockey, student council, rock band, ...)
3. Define **at least 3 classes**, one for each **role** on the team/organization.
4. Choose **2-3 attributes** and **2-3 behaviors** the roles typically exhibit.
5. Define a **superclass** of which each role you defined is a **subclass**.
6. Abstract **common attributes and behaviors** in the superclass that all subclasses will **inherit**.
7. Distinguish **attributes and behaviors that differ between roles** and are included only in the **subclasses** (not in the superclass).

Create a *Student* Subclass

Let's create a Student class that inherits from person!



```
class Student(Person):
  pass
```

Instantiating A Subclass

Even though our subclass doesn't contain any new properties and methods, it **inherits** all properties and methods from its superclass!

Note that we still must set the object's **`.name`** property to instantiate it.

```
# TODO: Change the class name and argument to your name
Jane = Student("Jane")

Jane.say_hello()
```

Create Method in Subclass

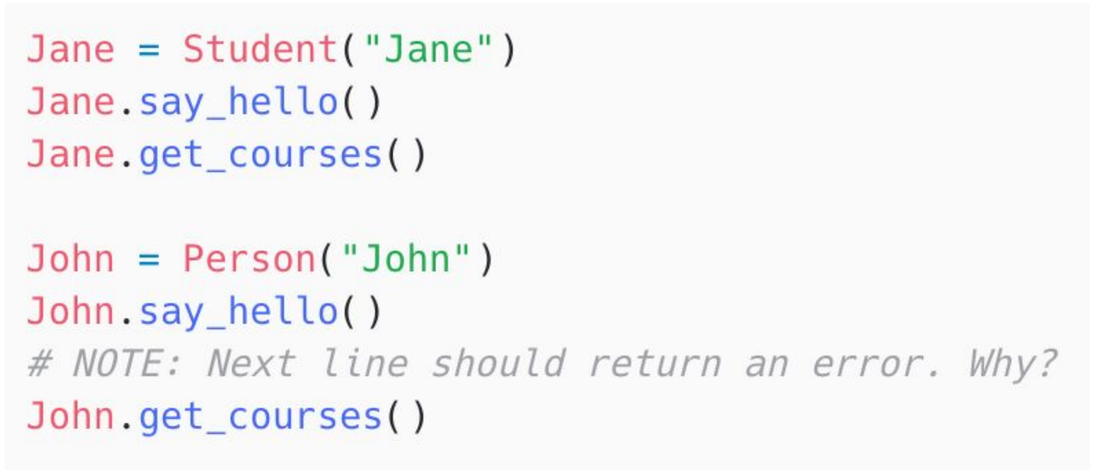
Properties and methods defined in our **subclass** do not live in our **superclass** and thus **are not inherited** by other subclasses. Let's write a method to print out your courses at Make School! Update your code to the following:

```python
class Student(Person):
  def get_courses(self):
    print("These are the current courses I'm taking: CS 1.1.")
```

Instantiating a Subclass and Superclass

Now that our subclass has inherited and explicitly defined methods, let’s test out both!

Run the following code to see the difference between superclass and subclass.



```python
Jane = Student("Jane")
Jane.say_hello()
Jane.get_courses()

John = Person("John")
John.say_hello()
# NOTE: Next line should return an error. Why?
John.get_courses()
```

Initializing Superclass in Subclass Initializer

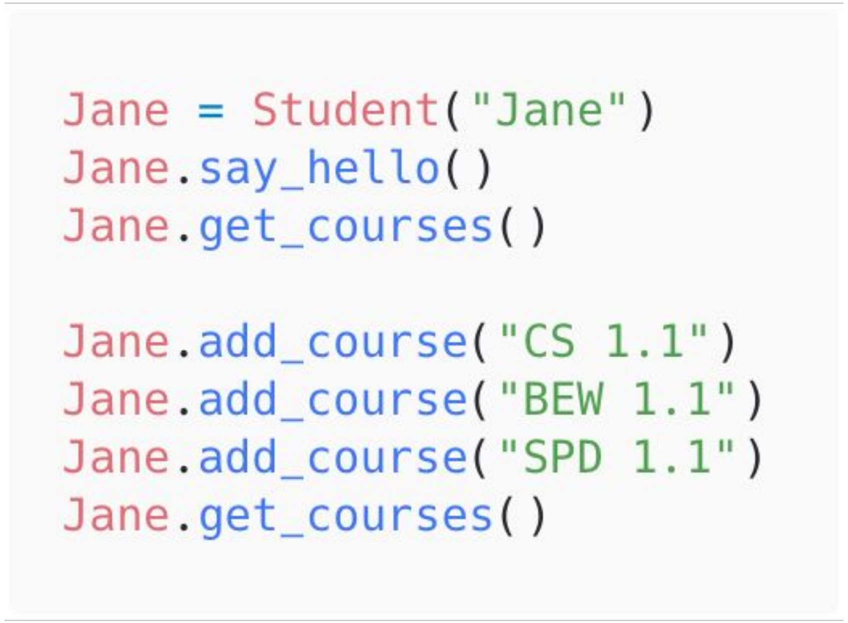
```python
class Student(Person):
  def __init__(self, name):
    # Initialize Person object with name
    super().__init__(name)
    # Student has not enrolled in courses
    self.courses = []

  def add_course(self, course):
    self.courses.append(course)

  def get_courses(self):
    count = len(self.courses)
    print(f"I'm taking {count} courses:")
    for course in self.courses:
      print(f"  - {course}")
```

Using New Improved Methods

Run the following code to see how we can use the new and improved methods:



```
Jane = Student("Jane")
Jane.say_hello()
Jane.get_courses()

Jane.add_course("CS 1.1")
Jane.add_course("BEW 1.1")
Jane.add_course("SPD 1.1")
Jane.get_courses()
```

Now it’s your turn!

Create a **subclass** of **`Person`** called **`Musician`**.

Create a method in the **`Musician`** class called **`.get_instruments()`** that prints the instruments that the **`Musician`** plays. Override the introduce_self() method to say name age and instruments

Instantiate a new **`Musician`** object that represents a musician that you like. 😁

Students browse: repl.it/@MakeSchool/createmusiciansubclass?lite=true

Awesome work!

You’ve successfully extended a class with properties and methods inherited from a superclass.

Summary

- **Objects** are structured collections of data in the form of variables (**properties**) and functions that act on those variables (**methods**).
- **Classes** are blueprint-like structures used to **create object instances**.
- To use a class, access its properties and call its methods, one must **instantiate an object** that represents an instance of that class.
- A **superclass** contains common properties and methods shared between several **subclasses** that **extend** and differentiate from their superclass.
- **Subclasses** will **inherit** all properties and methods from its **superclass**, which allows code using OOP to avoid duplication and remain DRY.

Virtualenv

MAKE SCHOOL

- Virtualenv is a tool used to create an isolated Python environment
- This environment has its own installation directories that doesn't share libraries with other virtualenv environments
- Why might using Python virtual environments be good engineering practice?

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Virtual Env demo

- pip3 install virtualenv
- virtualenv --version
- cd project_folder
- virtualenv venv
- source venv/bin/activate
- pip3 install ----
- deactivate
- pip3 freeze > requirements.txt
- pip3 install -r requirements.txt

VS Code Debugger demo

- Breakpoint
- Step Over
- Step Into
- Step Out
- Watch
- Call Stack

MAKE SCHOOL

Shout outs

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

OOP Worksheet!

Assignments

- Flower Garden
- Quiz 1 study guide given soon
- Next Time: Quiz 1 released Thursday
  - 30 min - 1 hr in gradescope
  - Open note open book
  - 3 days to complete, not timed
- Next Time: Superhero Team Dueler Tutorial