Object-Oriented Programming

OOP

By Ryan Henning

galvanize

- What / Why OOP?
- OOP Terminology
- OOP in Python
- Design Example
- Encapsulation
- Composition
- Inheritance
- Polymorphism

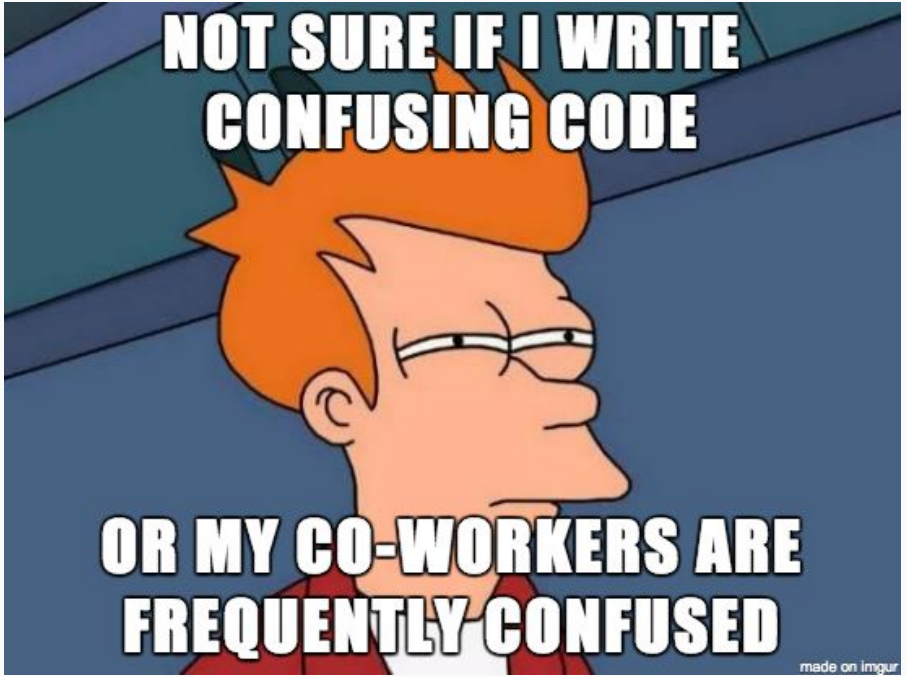
galvanize

Why OOP?

Software Engineering is the real topic today.

Goal is to write code that is:

- split into logical components
- easy to understand
- easy to use / re-use
- easy to modify
- easy to maintain
- easy to test



OOP is a programming paradigm

galvanize

What is a “paradigm”?

Paradigm: a distinct set of concepts or thought patterns

Other programming paradigms:

- **Functional:** no global state, deeply nested function calls
- **Imperative:** explicit sequence of commands (often called sequential)
- **Procedural:** like imperative, but also supports procedures (e.g., functions)
- **Declarative:** declare the result you want, not how to obtain it (e.g. SQL)
- …

Most programming languages offer a mix of paradigms.

What's in a program?

galvanize

"A program by any other name would smell as sweet." - Shakespeare's Juliet

Program = Data + Algorithm

For this presentation, we'll say:

Program = State + Behavior

galvanize

So, what is the OOP paradigm all about?

Group the program's state and behavior inside **objects** and **classes**.

Inspired by how humans categorize and manipulate the physical world.

E.g. Consider the concept of "Mug":

- Mug has state: color, volume, location
- Mug has behavior: drink, fill, refill, break, clean

What is the current state of this mug?

OOP Terminology: “Object”

An **object** is:

- A collection of variables (the “state” of the object)
- A collection of methods (the “behavior” of the object)

There are 5 “Mug” objects on this page. Each has distinct state, but the same behavior.

galvanize

OOP Terminology: “Class”

A **class** is a “blueprint” for objects. It’s the meta concept.

E.g. The previous slide showed 5 mug objects. Each of those 5 objects is an **instance** of the class: Mug. 5 objects, 1 class.

galvanize

How many objects? How many classes?

galvanize

OOP Terminology: “Member variable”

Every object has a set of **member variables**. Member variables are variables that are bound to that object and only that object. You can think of it like that object “owns” those variables.

An object’s member variables define that object’s current state.

galvanize

OOP Terminology: “Method”

Every object has a set of **methods**. Each method is a procedure that the object knows how to perform. Most of the time the methods will change the object’s state (i.e. modify the object’s member variables).

An object’s methods define that object’s behavior.

Usually an object’s methods are defined by the object’s class, however in Python you can give an object new methods at runtime. That’s why we say methods belong to the object, not to the class.

Live Python Demo: OOP in Python

Review:

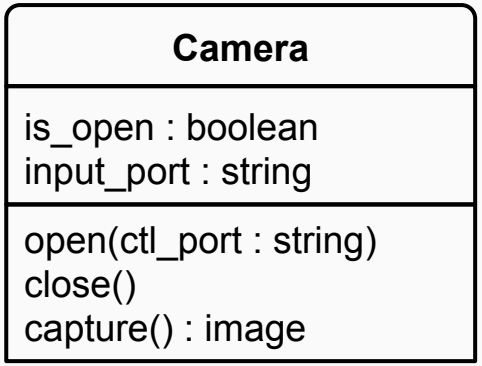
- Python classes you use every day! (list, dict, string, …)
- Instantiating Fraction objects
- Teach fractions how to add themselves together
- Magic methods

galvanize

UML Class Diagram

UML: Unified Modeling Language

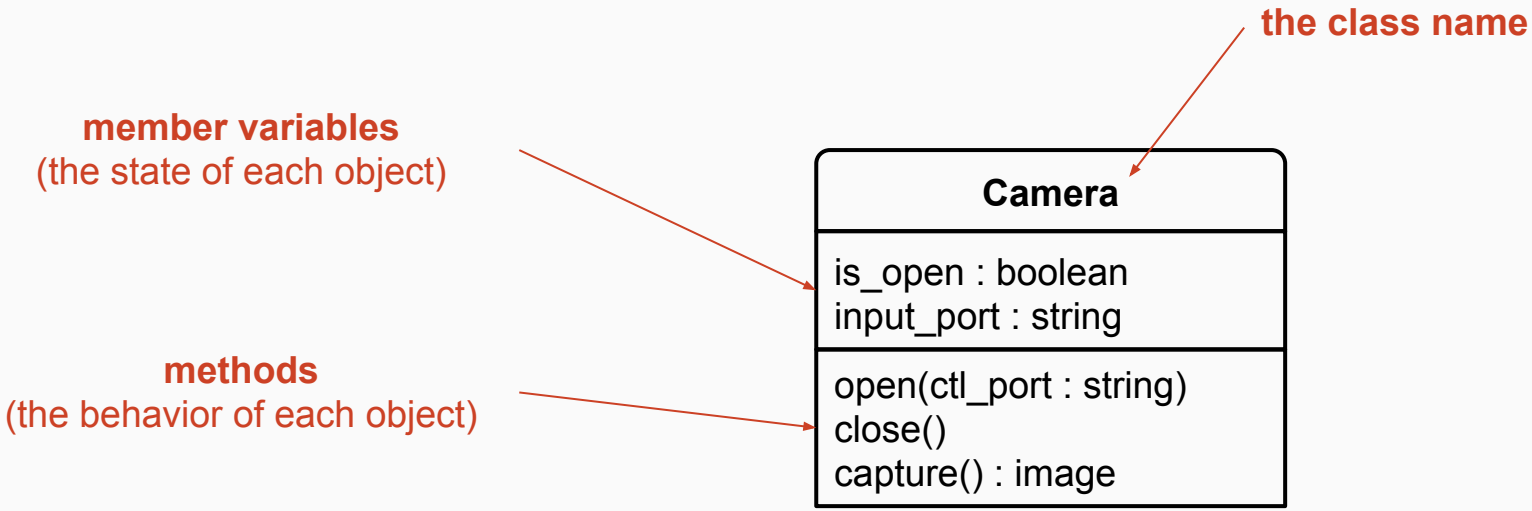
What are the parts of this class diagram?



galvanize

UML Class Diagram

UML: Unified Modeling Language



galvanize

Let’s design a program in OO fashion.

Smart Recycling Robot (SRR)

Build a robot that sorts physical objects into three categories:

1. recycle,
2. compost,
3. landfill

The robot brain is a computer program that you will write. The robot has an arm for grasping each piece of trash and delivering it to one of three bins.

galvanize

Software Engineering: Designing the SRR

Nouns and Verbs

Capture frames from the camera.

Process frames.

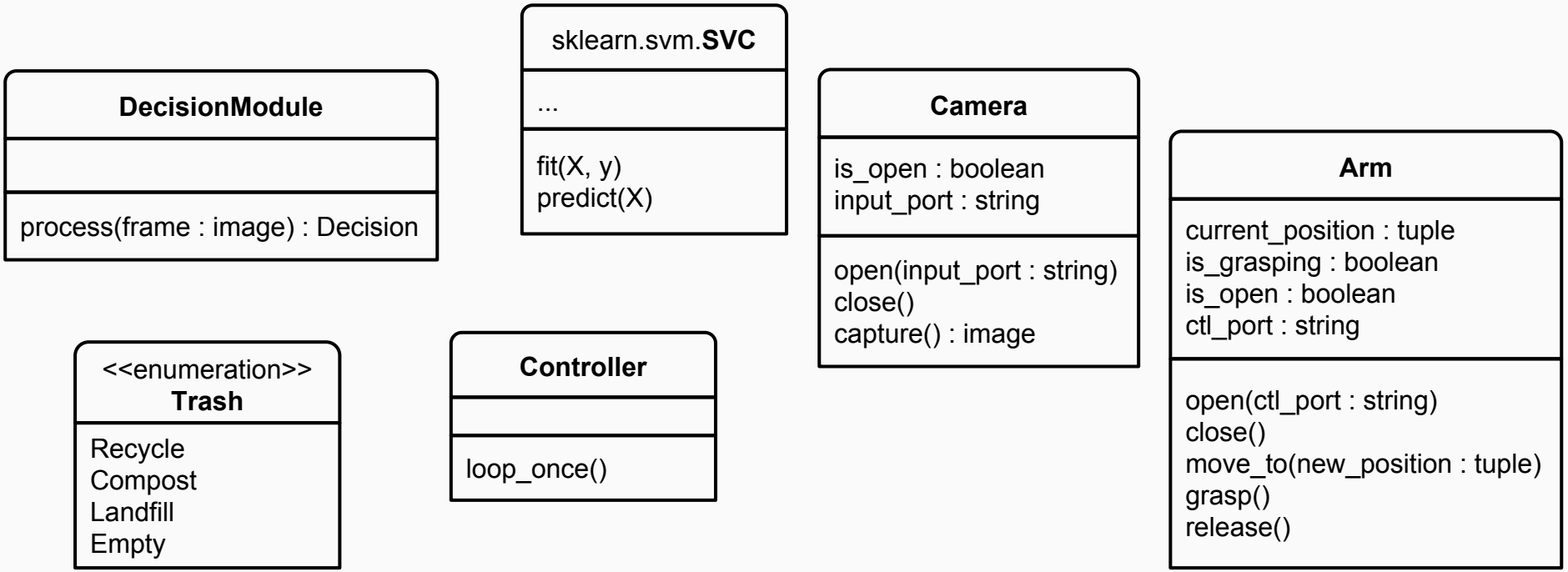
Move the arm into position.
Grasp the trash using the arm.
Release the trash.

Loop the detection/action cycle.

Types of trash are: Recycle, Compost, Landfill, Empty

Software Engineering: Designing the SRR

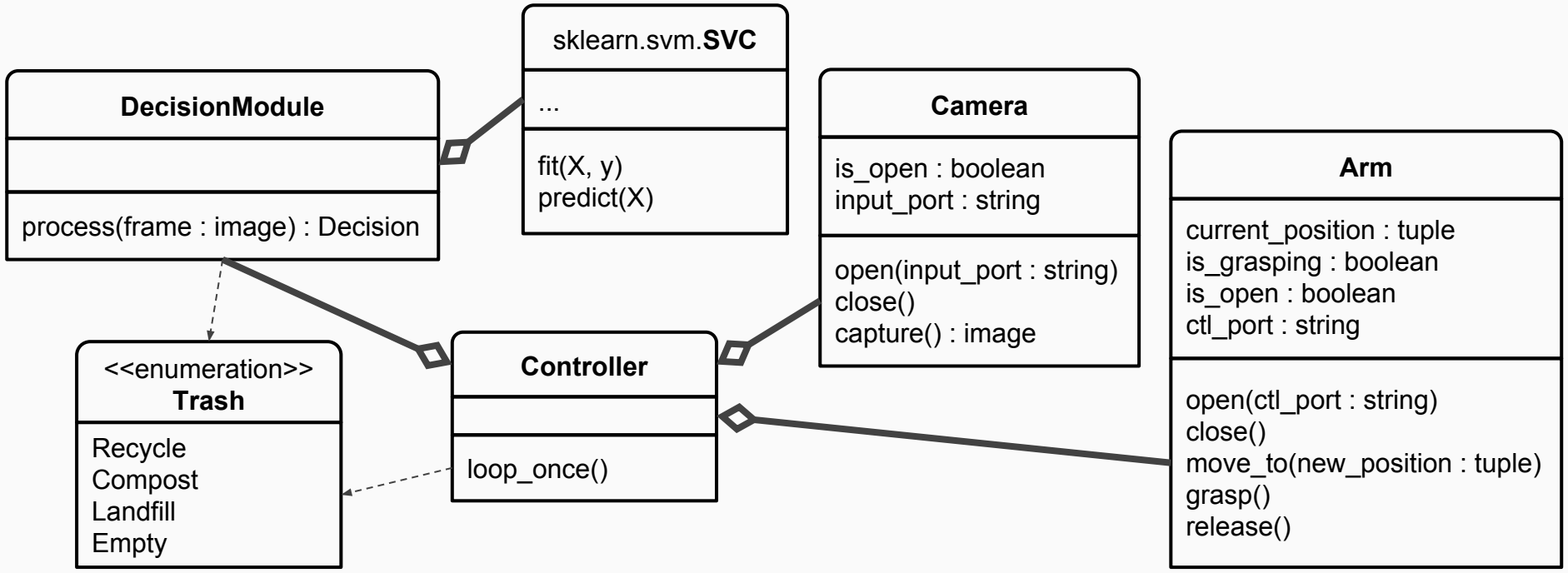
UML Class Diagram with Relationships



galvanize

Software Engineering: Designing the SRR

UML Class Diagram with Relationships



Live Python Demo: construct the SRR

OOP FTW in (at least) 3 ways: what are they?

galvanize

Encapsulation

Hiding the confusing details/complexity of your code inside a class.

Good because:

1. Code outside the class is safe (encouraged, even) to ignore the details inside the class.
2. The details/complexity inside the class are free to change without affecting the code outside the class.

Encapsulation (example)

```
>>> import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
>>> y = np.array([1, 1, 2, 2])
>>> from sklearn.svm import SVC
>>> clf = SVC()
>>> clf.fit(X, y)
>>> print(clf.predict([[-0.8, -1]]))
[1]
```

Composition

Storing objects inside objects. The “has-a” relationship. Triangle has Points.

```
class Point:
    ... details omitted ...
    def calculate_distance(self, other_point):
        ... details omitted ...


if __name__ == '__main__':
    p1 = Point(0, 2)
    p2 = Point(4, 5)
    p3 = Point(3, 0)
    triangle = Traingle(p1, p2, p3)
    print triangle.calculate_perimeter()
```

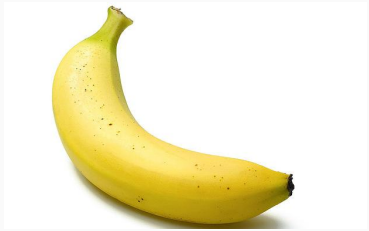
```
class Triangle:
    def __init__(point1, point2, point3):
        self.point1 = point1
        self.point2 = point2
        self.point3 = point3

    def calculate_perimeter(self):
        a, b, c = self.point1, self.point2, self.point3
        return a.calculate_distance(b) +
               b.calculate_distance(c) +
               c.calculate_distance(a)
```

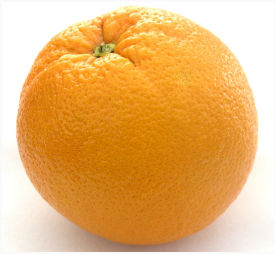
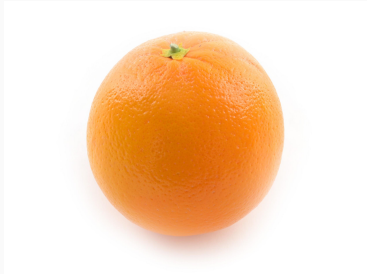
Morning Exercise

Modify an existing OOP-style program: The (card)game of WAR.

galvanize

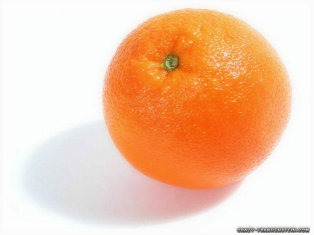
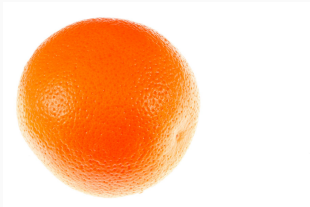
How many objects? How many classes?





Inheritance

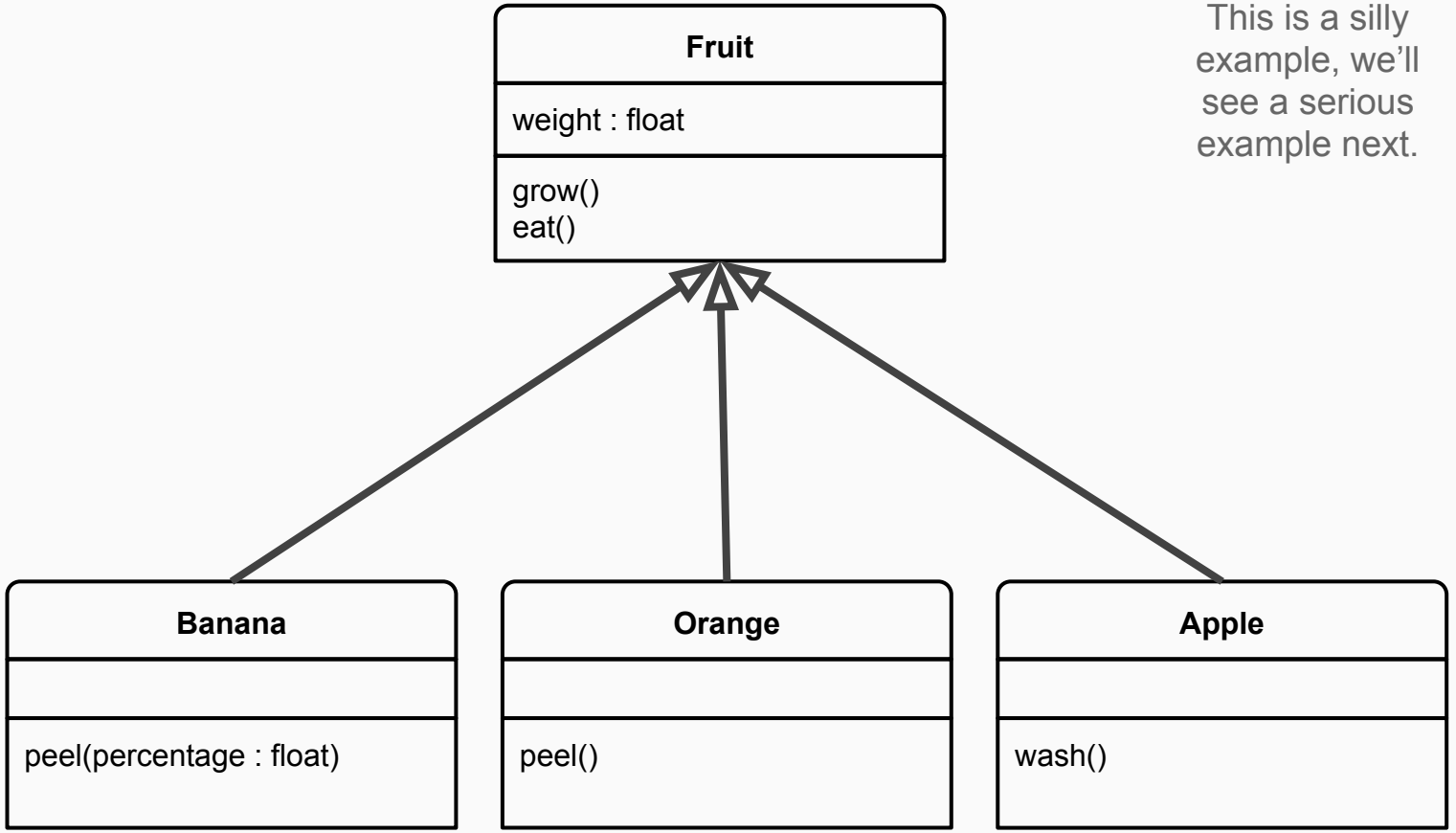
The “is-a” relationship. Banana is a Fruit.

galvanize

This is a silly example, we’ll see a serious example next.

“Parent” class:

“Child” classes:

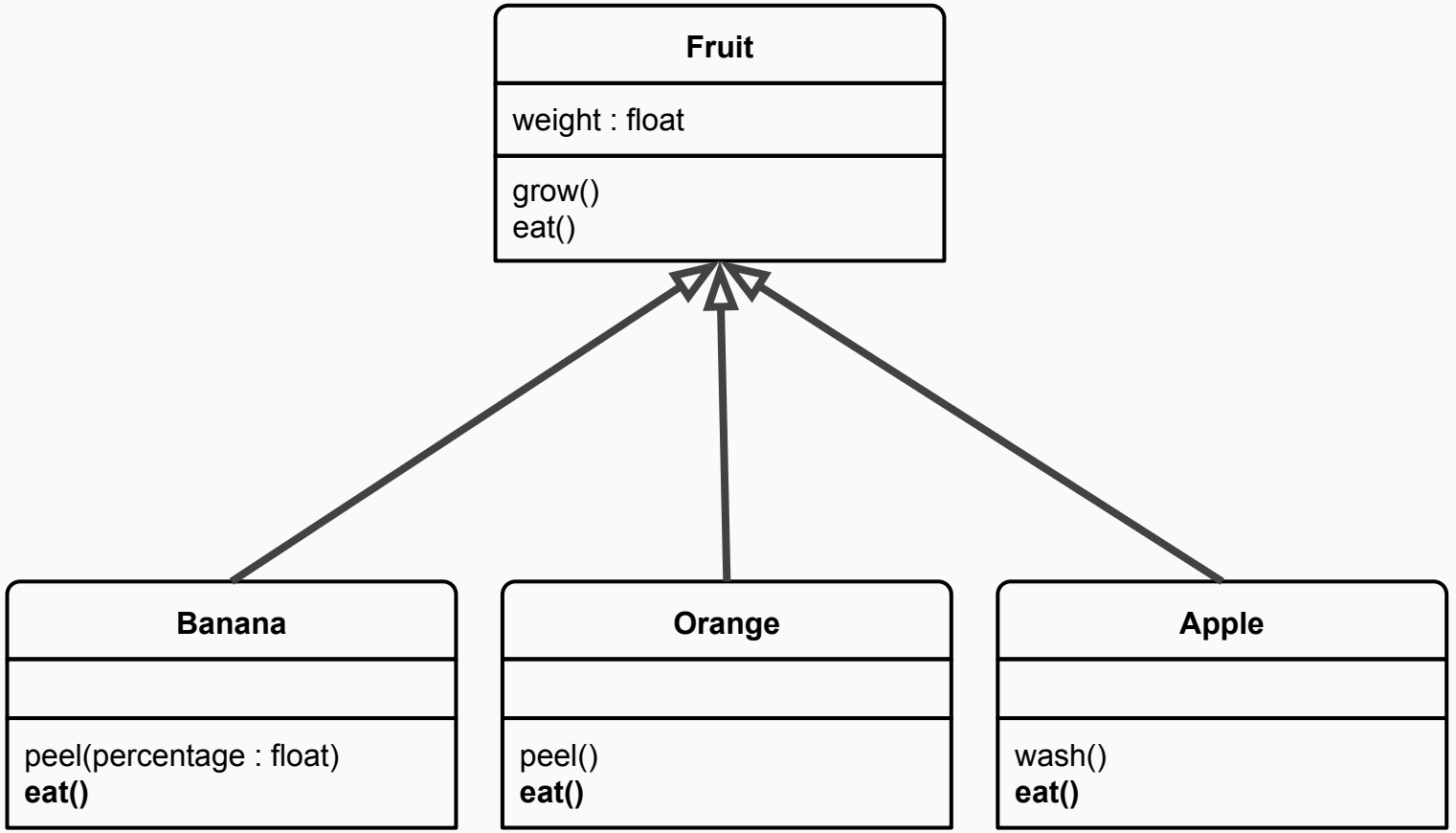


Polymorphism

galvanize

“Parent” class:

“Child” classes:

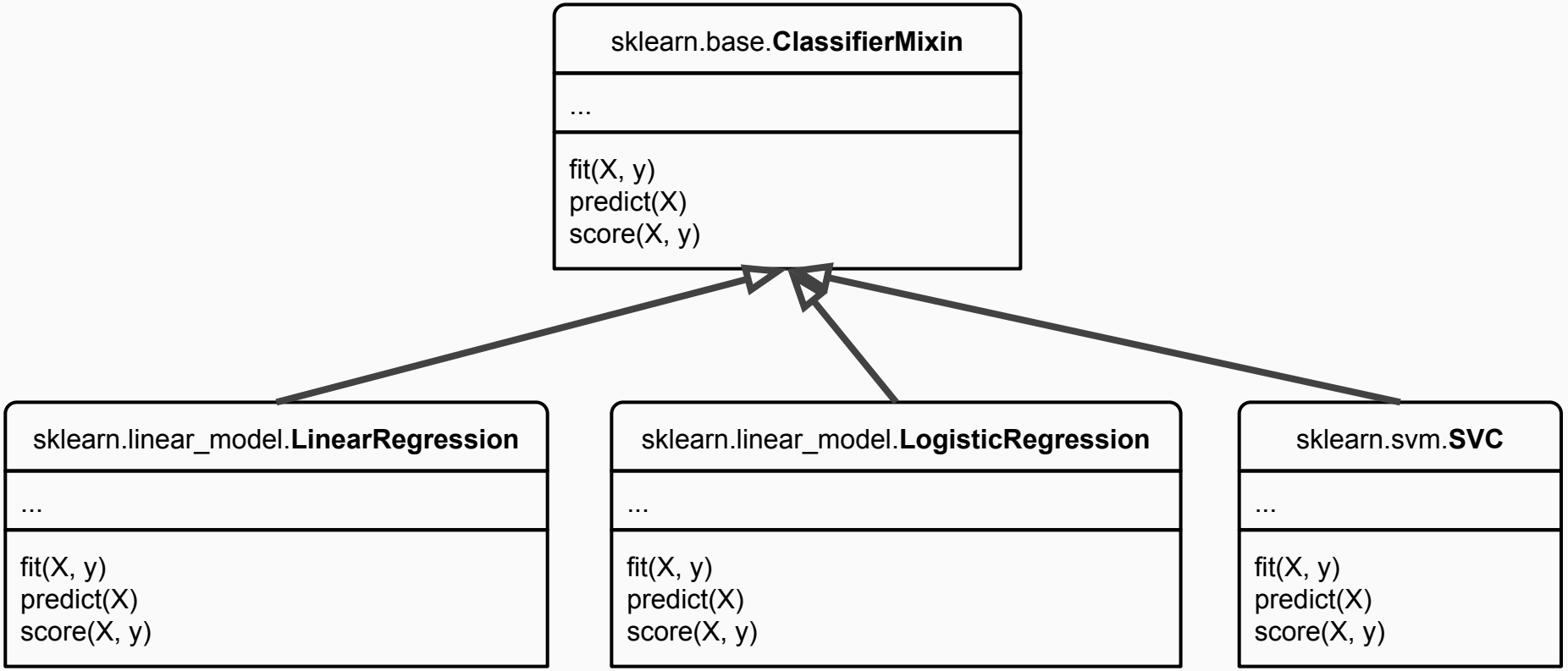


Live Python Demo: Inheritance & Polymorphism

Fruit / Banana / Orange / Apple

Polymorphism in sci-kit learn

galvanize



Notes about Python convention

Classes:

- Class names are nouns.
- Class names use *UpperCamelCase*.

Methods:

- Method names are (usually) verbs.
- Method names use *snake_case*.
- Method names that begin with an underscore should be treated as “private”.

Live Python Demo:
Testing code via mock objects

Testing the Controller.

Afternoon Exercise

Simulate the game of Blackjack in an OOP-style program.