Object Oriented Programming

Objectives

After this lecture you will be able to:

- Define OOP (Object oriented programming)
- Explain why we use it
- Define important terms associated with objects and classes in Python
- Write your own classes to make your own objects
  - Understand Python class syntax
- Use magic methods to add functionality to your objects
- Provide a rationale of when to use classes or functions
- Define Encapsulation, Inheritance, Polymorphism

What is OOP?

Object-oriented programming (OOP) is a **programming paradigm** based on the concept of **objects.**

In Python, **objects** are data structures that contain **data**, known as **attributes**; and **procedures**, known as **methods**.

Why OOP (1)?

OOP was developed to :

- Help build large-scale software
- Promote software reuse (keep well tested code)
- Decouple code; improve maintenance and stability of code
- Hide details away from those that use the code.

Early OOP:

```
Class FittingRoom; Begin
      Ref (Head) door;
      Boolean inUse;
      Procedure request; Begin
         If inUse Then Begin
             Wait (door);
             door.First.Out;
         End;
         inUse:= True;
      End;
      Procedure leave; Begin
         inUse:= False;
         Activate door.First;
      End;
      door:- New Head;
   End;
```

Simula 67, developed in Oslo, Norway in 1967 and based on the ALGOL 60 programming language
https://en.wikipedia.org/wiki/Simula

Why OOP (1)?

The FittingRoom class defines (it's the blueprint for) how FittingRoom objects are made.

What attributes (data) will a FittingRoom object have?

What methods (procedures)?

Early OOP:

```
Class FittingRoom; Begin
      Ref (Head) door;
      Boolean inUse;
      Procedure request; Begin
         If inUse Then Begin
             Wait (door);
             door.First.Out;
         End;
         inUse:= True;
      End;
      Procedure leave; Begin
         inUse:= False;
         Activate door.First;
      End;
      door:- New Head;
   End;
```

Simula 67, developed in Oslo, Norway in 1967 and based on the ALGOL 60 programming language https://en.wikipedia.org/wiki/Simula

Why OOP (2)?

As people, we are used to the idea of objects, and it isn't difficult to make the leap from characteristics of physical objects to those of programming objects.

What characteristics could you use to describe this cup?
(What are its attributes?)

What can you do with this cup?
(*What are its methods?*)

You've been using Python objects:

```
In [4]: cards = ['Ace_of_hearts','8_of_Diamonds','4_of_Spades']
```

The cards object is created (*instantiated*) with data (*attributes*).

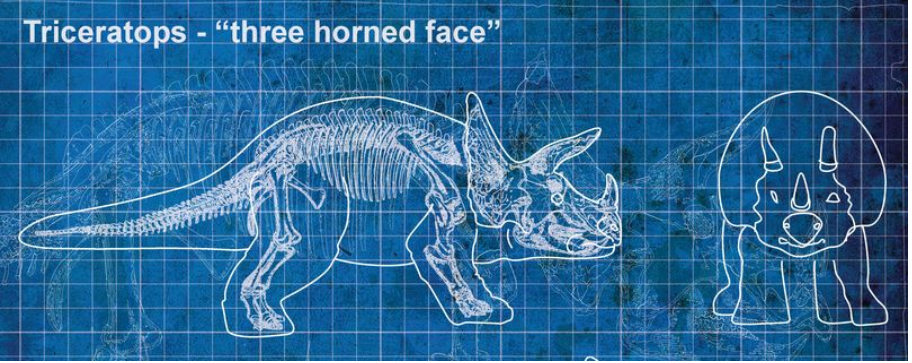
What type of object is cards?

```
In [5]: type(cards)
Out[5]: list
```

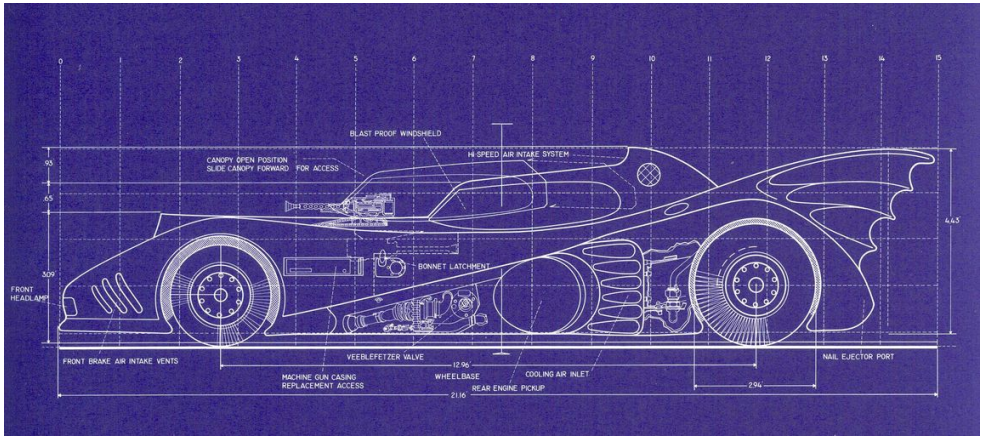
cards is a *list object* based on the *list class*.

What methods are available to do things with data in cards?

```
In [6]: cards.
        cards.append  cards.index   cards.remove
        cards.count   cards.insert  cards.reverse
        cards.extend  cards.pop     cards.sort
```


Triceratops - “three horned face”

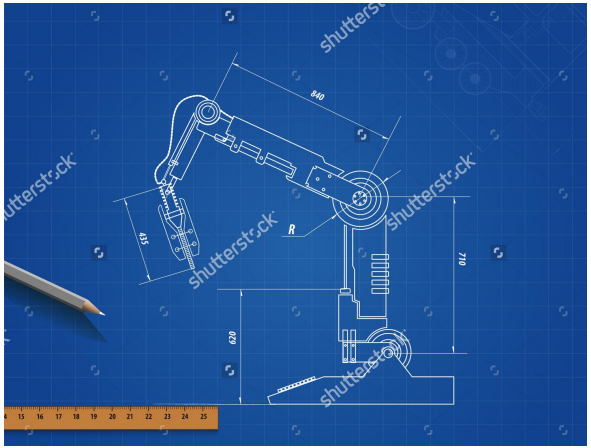
Classes


BLAST PROOF WINDSHIELD
HI SPEED AIR INTAKE SYSTEM
CANOPY OPEN POSITION
SLIDE CANOPY FORWARD FOR ACCESS
BONNET LATCHMENT
FRONT HEADLAMP
FRONT BRAKE AIR INTAKE VENTS
VEEBLEFETZER VALVE
MACHINE GUN CASING REPLACEMENT ACCESS
WHEELBASE
COOLING AIR INLET
REAR ENGINE PICKUP
NAIL EJECTOR PORT

How to define your own objects: classes

A **class** is a blueprint that describes the format of an object. It tells us what *attributes* an object will store , and what *methods* that object will have available. The class defines how an object is built.

One of the goals of today is to show you how to write classes so that you can build your own objects to store data and operate on that data as you wish.



Quick quiz

How many objects?
How many classes?

How to write classes in Python (class syntax)

```
class CamelCaseObjectName(object):
    '''Document string describing what class does'''

    def __init__(self, parameter1, parameter2):
        self.parameter1 = parameter1
        self.parameter2 = parameter2
        self.otherstuff = self.method1()

    def method1(self):
        return self.parameter1 * self.parameter2
```

Class syntax

```
class CamelCaseObjectName(object):
    '''Document string describing what class does'''

    def __init__(self, parameter1, parameter2):
        self.parameter1 = parameter1
        self.parameter2 = parameter2
        self.otherstuff = self.method1()

    def method1(self):
        return self.parameter1 * self.parameter2
```

__init__ runs when object instantiated

attributes

method to do something with attributes

method runs when called

`self` allows the object to refer to its own attributes and methods

Magic methods

Special methods, indicated by double underscore, that you can use to give ubiquitous functionality of some operators to objects defined by your class.

A sampling:

Magic method	Purpose
`__init__(self, [..)`	`Constructor, initializes the class`
`__repr__(self)`	`Defines format for how object should be represented`
`__len__(self)`	`Return number of elements in an object`
`__gt__(self, other)`	`Implements greater than operator, >`
`__add__(self, other)`	`Implements addition, +`

For a nice Python magic method reference see:
https://github.com/RafeKettler/magicmethods/blob/master/magicmethods.markdown

Programming example

A Fraction class

$$\frac{1}{2} \qquad \frac{2}{3} + \frac{1}{4} = \frac{11}{12}$$

Fraction class attributes?

Fraction class methods?

See `fraction.py` code

In-class exercise

Write a class (`Die.py`) to make an n-sided die

After a die is instantiated let the user be able to query:

- How many sides it has
- What number is face up (its value)

Also, let the user be able to:

- Roll the die
- Compare the values of two die (>, <, ==, >=, <=)

Think about it, write a python script, test it, then Slack it to a colleague in class to check!

Use *class* or *function*?

No hard rules, but rule-of-thumb:

If can think of what you're doing as a noun, use a class.

If you can think of it as a verb, use a function.
(Or, a method within a class!).

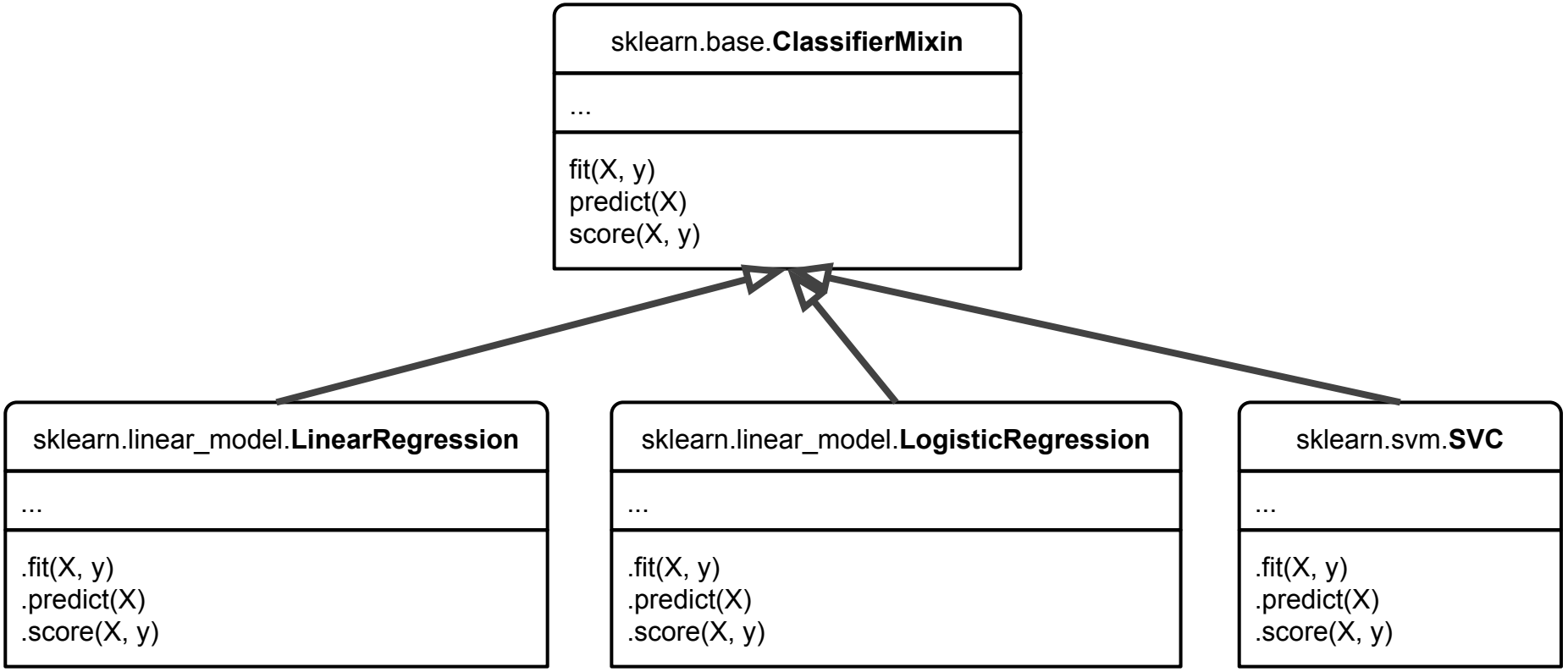
Advantages of OOP

- **Inheritance** - When a class is based on another class, building off of the existing class to take advantage of existing behavior, while having additional specific behavior of its own.
  Example: class Animal, subclasses Mammal, Bird, Crustacean, etc
- **Encapsulation** - The practice of hiding the inner workings of our class, and only exposing what is necessary to the outside world. This idea is effectively the same as the idea of **abstraction**, and allows users of our classes to only care about the what (i.e. what our class can do) and not the how (i.e. how our class does what it does).
  Example: cards.reverse()
- **Polymorphism** - The provision of a single interface to entities of different types. This enables us to use a shared interface for similar classes while at the same time still allowing each class to have its own specialized behavior.
  Example: integers and floats are implicitly polymorphic since you can add, subtract, multiply and so on, irrespective of the fact that the types are different.

Inheritance example: `box.py`

Encapsulation example: `deck.py`

Polymorphism (and how it benefits you as a DS)



Quick review

What is the difference between an *object* and a *class*?

What is the difference between an *attribute* and a *method*?

What is the syntactic difference between an *attribute* and a *method*?

What is the role of *self* in defining a *class*?

What can be used to give a custom class functionality similar to other classes?

How can we see the *attributes* and *methods* available on an *object* in IPython?

How do you decide when to use a *class* or when to use a *function*?

Objectives

After this lecture you will be able to:

- Define OOP (Object oriented programming)
- Explain why we use it
- Define important terms associated with objects and classes in Python
- Write your own classes to make your own objects
  - Understand Python class syntax
- Use magic methods to add functionality to your objects
- Provide a rationale of when to use classes or functions
- Define Encapsulation, Inheritance, Polymorphism