Motivation

- Physically, items often share common aspects
- Example:
  - cars, semis, motorcycles are all road vehicles
  - all have wheels (different #s)
  - some have windows, ...

Motivation

- Coding each separately would repeat any common aspects
- Logically different -> makes sense to code as different things
- How can we represent and capitalize on logical relationship?

Inheritance - Basic Idea

- Represent connections between classes via inheritance
- parent/super/base class
- child/sub class
- allows for code/software reuse

Is-a vs has-a

- "has-a" relationship represents variables in classes
- "is-a" relationship signals potential child
- If X is derived from Y, should be able to say X is a Y

is-a vs has-a

- Example:
  - mustang is a car
  - car is a road vehicle
  - car has windows, trunk
- Example:
  - horse is a mammal
  - dog is a mammal

Abstract Classes

- Sometimes parent classes are for organization and structure
- May want to represent higher level thing that isn't actually an object we want to instantiate
- Example 1:
  - Parent class = Animal
  - Child classes = Dog, Cat, Lion
- Example 2:
  - Parent class = Bicycle
  - Child class = MountainBike, ElectricBike, RoadBike, TriBike

Abstract Classes

- Solution: abstract class

```
from abc import ABC, abstractmethod

class Animal(ABC):
    def __init__(self):
        pass
```

Abstract Classes

- Might also see

```
from abc import ABCMeta, abstractmethod

class Animal(metaclass=ABCMeta):

    def __init__(self):
        pass
```

Abstract Methods

- Method without implementation (body)
- Used in abstract classes
- Forces child classes to implement
- Declare with a decorator

Abstract Methods

```
from abc import ABC, abstractmethod

class Animal(ABC):
    def __init__(self):
        pass

    @abstractmethod
    def move(self, changeX, changeY):
        pass
```

Abstract Classes (cont.)

- abstract classes with abstract methods:
  - cannot be instantiated
  - have no practical use until they have a descendent class
- If child does not implement all `abstract` methods -> child is also abstract
- Can have concrete methods as well
- Can still have constructors - typically called in child constructor