



UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science



Object-Oriented Programming: Part 2, Inheritance



Announcements

- Midterm 10/ 7-9pm
 - Locations and assignments will be sent early next week
 - Unlimited Handwritten Sheets – but try to use no more than 3-4!
- Lecture: Thurs 10/14 – More OOP Practice
- Lecture: Tues 10/18 – Exam Review / Q&A
- Check the Calendar!
 - Exam Review Sessions led by Tutors Fri 3-5pm (Cory 277) (time moved!)
 - New “topical” review sessions by TAs
 - CSM review sessions too
- No labs next week
- TAKE A DEEP BREATH! Y’all can do this. 😊



Classes Can Have Attributes Too!

- Class attributes (as opposed to *instance* attributes) belong to the class itself, instead of each object
 - This means there is one value which is shared for all of the class's objects
- Be Careful!
 - It's easy to overdo class attributes
- Methods that rely only on class attributes are called *class methods*
 - Python has some special features we won't use, but are useful:
 - >> <https://docs.python.org/3/library/functions.html?highlight=classmethod#classmethod>



Example: class attribute

```
class BaseAccount:
    account_number_seed = 1000

    def __init__(self, name, initial_deposit):
        self._name = name
        self._balance = initial_deposit
        self._acct_no = BaseAccount.account_number_seed
        BaseAccount.account_number_seed += 1

    def name(self):
        return self._name

    def balance(self):
        return self._balance

    def withdraw(self, amount):
        self._balance -= amount
        return self._balance
```



More class attributes

```
class BaseAccount:
    account_number_seed = 1000
    accounts = []
    def __init__(self, name, initial_deposit):
        self._name = name
        self._balance = initial_deposit
        self._acct_no = BaseAccount.account_number_seed
        BaseAccount.account_number_seed += 1
        BaseAccount.accounts.append(self)

    def name(self):
        ...

    def show_accounts():
        for account in BaseAccount.accounts:
            print(account.name(),
                  account.account_no(), account.balance())
```



UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science



Object-Oriented Programming: Inheritance



Learning Objectives

- Inheritance allows classes to reuse methods and attributes from a parent class.
- `super()` is a new method in Python
- Subclasses or child classes are distinct from one another, but share properties of the parent.



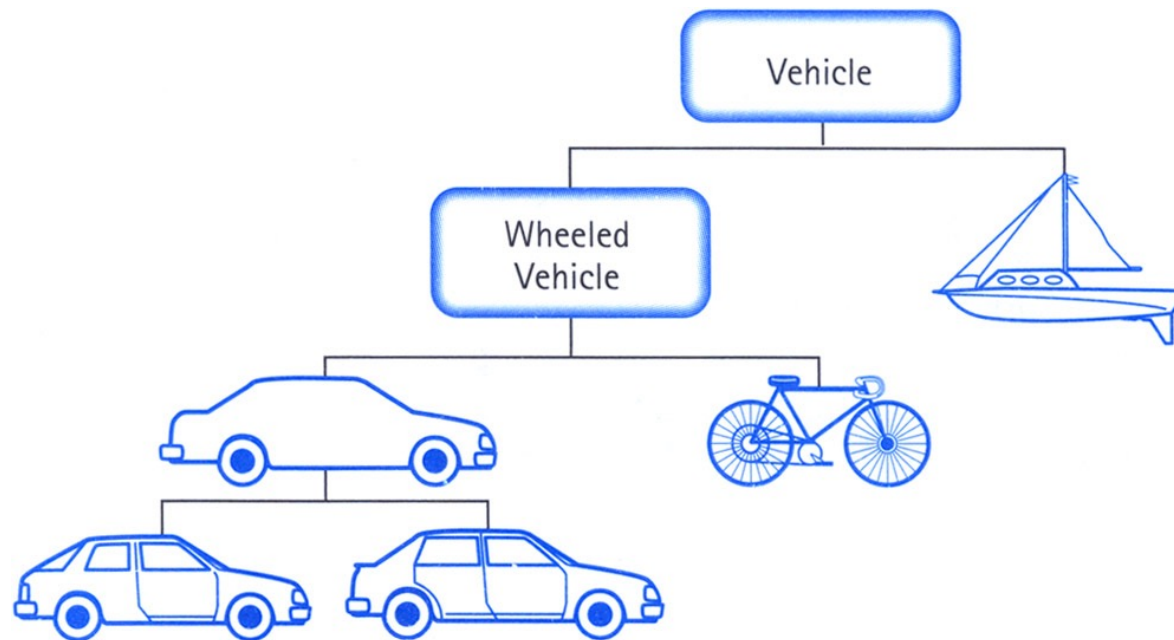
Inheritance

- Define a class as a specialization of an existing class
- Inherent its attributes, methods (behaviors)
- Add additional ones
- Redefine (specialize) existing ones
 - Ones in superclass still accessible in its namespace



Class Inheritance

- Classes can inherit methods and attributes from parent classes but extend into their own class.





Python class statement

```
class ClassName:  
    <statement-1>  
    .  
    .  
    .  
    <statement-N>
```

```
class ClassName ( inherits / parent-class ):  
    <statement-1>  
    .  
    .  
    .  
    <statement-N>
```



Example

```
class BaseAccount:
    def __init__(self, name, initial_deposit):
        # Initialize the instance attributes
        self._name = name
        self._acct_no = Account._account_number_seed
        Account._account_number_seed += 1
        self._balance = initial_deposit

class CheckingAccount(BaseAccount):
    def __init__(self, name, initial_deposit):
        # Use superclass initializer
        BaseAccount.__init__(self, name, initial_deposit)
        # Alternatively:
        # super().__init__(name, initial_deposit)
        # Additional initialization
        self._type = "Checking"
```



Accessing the Parent Class

- `super()` gives us access to methods in the parent or “superclass”
 - Can be called anywhere in our class
 - Handles passing `self` to the method
 - Handles looking up an attribute on a parent class, too.
- We can directly call `ParentClass.method(self, ...)`
 - This is not quite as flexible if our class structure changes.
- In general, prefer using `super()`!
- Outside of C88C, things can get complex...
 - <https://docs.python.org/3/library/functions.html#super>



UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science



Object-Oriented Programming: Evolving The Bank Model



Composing Classes Together

- Currently, our BaseAccount stores a lot of data in class attributes...
- This suggests we are trying to accomplish an entirely new kind of class, or object
 - A Bank!
- We should extract that these functions into their own class
- A bank can now manage:
 - making accounts
 - keeping track of account numbers
 - showing and listing accounts

Live Demo

