



UC Berkeley EECS  
Lecturer  
Michael Ball

# Computational Structures in Data Science

---



## Object-Oriented Programming: Part 2, Inheritance



## Announcements

---

- Midterm 3/16 7-9pm
  - Remote option available
  - Please wear masks in the exam room
  - Unlimited Handwritten Sheets – but try to use no more than 3-4!
- Lecture: Monday 3/14 – Review/Q&A
- Lecture: Weds 3/16 – No Lecture, just study / relax.
- then you've got Spring break!



UC Berkeley EECS  
Lecturer  
Michael Ball

# Computational Structures in Data Science

---



## Object-Oriented Programming: "Magic" Methods



## Learning Objectives

---

- Python's Special Methods define built-in properties
  - `__init__` # Called when making a new instance
  - `__sub__` # Maps to the `-` operator
  - `__str__` # Called when we call `print()`
  - `__repr__` # Called in the interpreter



## Special Initialization Method

---

`__init__` is called automatically when we write:

```
my_account = BaseAccount('me', 0)
```

```
class BaseAccount:

    def __init__(self, name, initial_deposit):
        self.name = name
        self.balance = initial_deposit

    def account_name(self):
        return self.name

    def account_balance(self):
        return self.balance

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



## More special methods

---

```
class BaseAccount:
    ... (init, etc removed)
    def deposit(self, amount):
        self._balance += amount
        return self._balance

    def __repr__(self):
        return '< ' + str(self._acct_no) +
            '[' + str(self._name) + ']' >'

    def __str__(self):
        return 'Account: ' + str(self._acct_no) +
            '[' + str(self._name) + ']'

    def show_accounts():
        for account in BaseAccount.accounts:
            print(account)
```



## More Magic Methods

---

- We will **not** go through an exhaustive list!
- Magic Methods start and end with “double underscores” `__`
- They map to built-in functionality in Python. Many are logical names:
  - `__add__` => + operator
  - `__sub__` => - operator
  - `__getitem__` => `[]` operator
- A longer list for the curious:
  - <https://docs.python.org/3/reference/datamodel.html>

# Live Demo

---







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

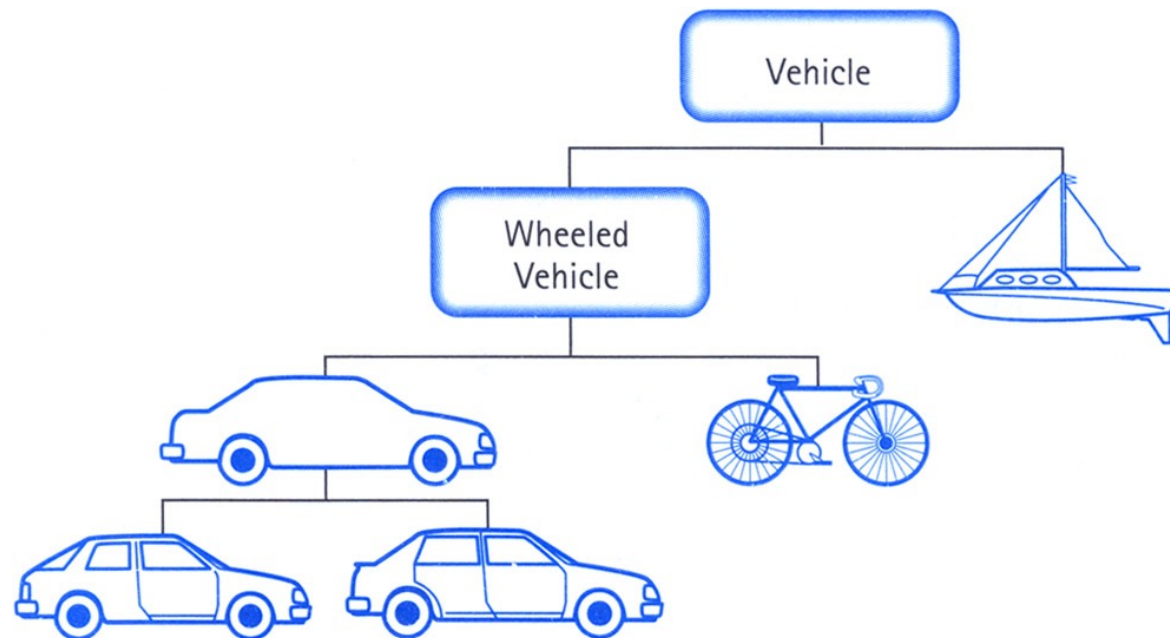
---

- 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 Account:
    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(Account):
    def __init__(self, name, initial_deposit):
        # Use superclass initializer
        Account.__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
- We can directly call `ParentClass.method(self, ...)`
  - This is not quite as flexible if our class structure changes.