



UC Berkeley  
EECS  
Lecturer  
Michael Ball

# Computational Structures in Data Science

---



## 13: Object-Oriented Programming



## Announcements

---

- Midterm Weds 3/16, 7-9pm
- Reference Sheet linked on the website, will be making some updates to this soon.



UC Berkeley  
EECS  
Lecturer  
Michael Ball

# Computational Structures in Data Science

---



## Object-Oriented Programming



## Learning Objectives

---

- Learn how to make a class in Python
  - Class keyword
  - `__init__` method
  - `self`



# Object-Oriented Programming (OOP)

- **Objects as data structures**

- With **methods** you ask of them

- »These are the behaviors

- With **local state**, to remember

- »These are the attributes

- **Classes & Instances**

- Instance an example of class

- E.g., Fluffy is instance of Dog

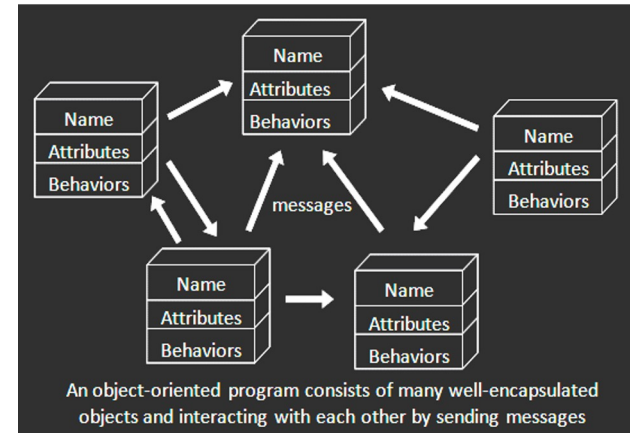
- **Inheritance saves code**

- Hierarchical classes

- E.g., pianist special case of musician,  
a special case of performer

- **Examples (though not pure)**

- Java, C++

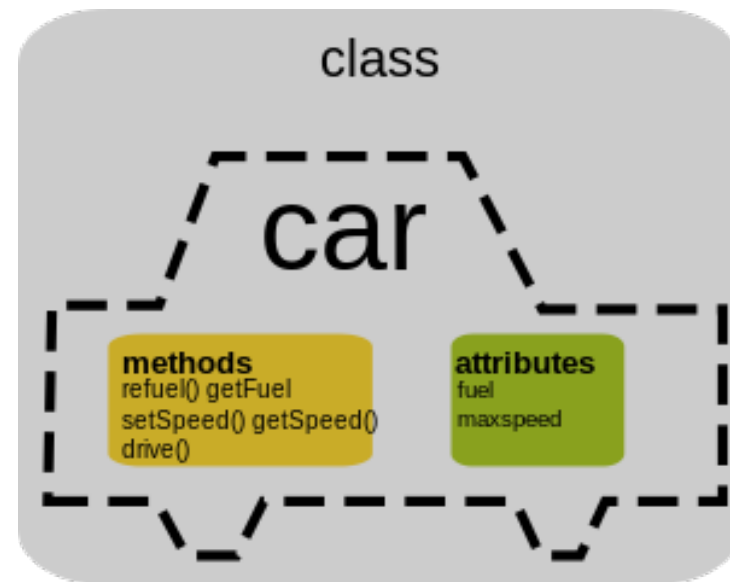


[www3.ntu.edu.sg/home/ehchua/programming/java/images/OOP-Objects.gif](http://www3.ntu.edu.sg/home/ehchua/programming/java/images/OOP-Objects.gif)



# Classes

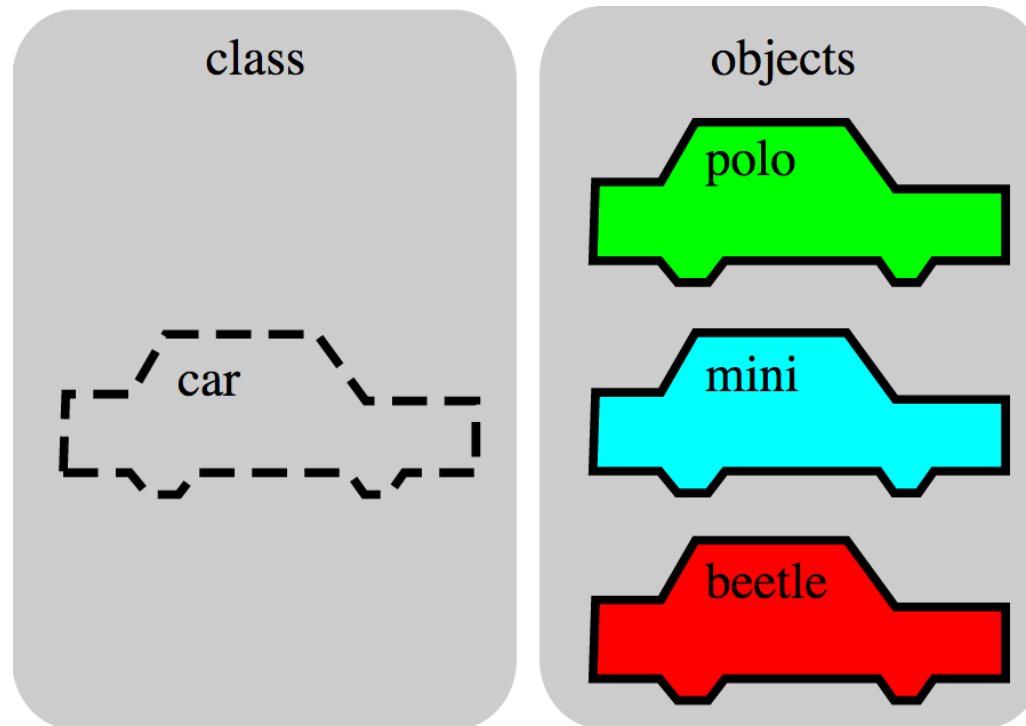
- Consist of data and behavior, bundled together to create abstractions
  - Abstract Data Types
- A class has
  - attributes (variables)
  - methods (functions)that define its behavior.



# Objects



- An object is the instance of a class.





# Objects

---

- Objects are concrete instances of classes in memory.
- They can have state
  - mutable vs immutable (lists vs tuples)
- Functions do one thing (well)
  - Objects do a collection of related things
- In Python, everything is an object
  - All **objects** have **attributes**
  - Manipulation happens through **methods**





## Python class statement

---

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

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



## Example: Account

```
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
```

new namespace

attributes

The object

dot

methods



## Creating an object, invoking a method

---

The Class Constructor

```
my_acct = BaseAccount("John Doe", 93)  
my_acct.withdraw(42)
```

dot



## Special Initialization Method

---

```
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
```

*return None*



## More on Attributes

---

- Attributes of an object accessible with ‘dot’ notation  
`obj.attr`
- You can distinguish between “public” and “private” data.
  - Used to clarify to programmers how you class should be used.
  - In Python an `_` prefix means “this thing is private”
  - `_foo` and `__foo` do different things inside a class.
  - [More for the curious.](#)
- Class variables vs Instance variables:
  - Class variable set for all instances at once
  - Instance variables per instance value



## Example

---

```
class BaseAccount:

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

    def name(self):
        return self.name

    def balance(self):
        return self.balance

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



## Example: “private” attributes

---

```
class BaseAccount:

    def __init__(self, name, initial_deposit):
        self._name = name
        self._balance = initial_deposit

    def name(self):
        return self._name

    def balance(self):
        return self._balance

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



## 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())
```