Computational Structures in Data Science

Object-Oriented Programming

Berkeley
UNIVERSITY OF CALIFORNIA

Announcements

- Midterm Scores out Soon
  - Finishing grading
  - Will open regrade requests ~48 hours after grades are released
  - Regrade Requests will be accepted for 1 week
- Midterm Clobber Policy:
  - If your final score improves your midterm score improves
  - e.g. 60% on MT, and 70% on Final, becomes a 70% MT score automatically
- Reminder: Lab 6 / HW 6 due this week
- Lab 7 and HW 7 released this week, due next week.

Computational Structures in Data Science

Object-Oriented Programming

Berkeley
UNIVERSITY OF CALIFORNIA

Learning Objectives

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

Object-Oriented Programming (OOP)

- **<u>Objects</u>** as data structures
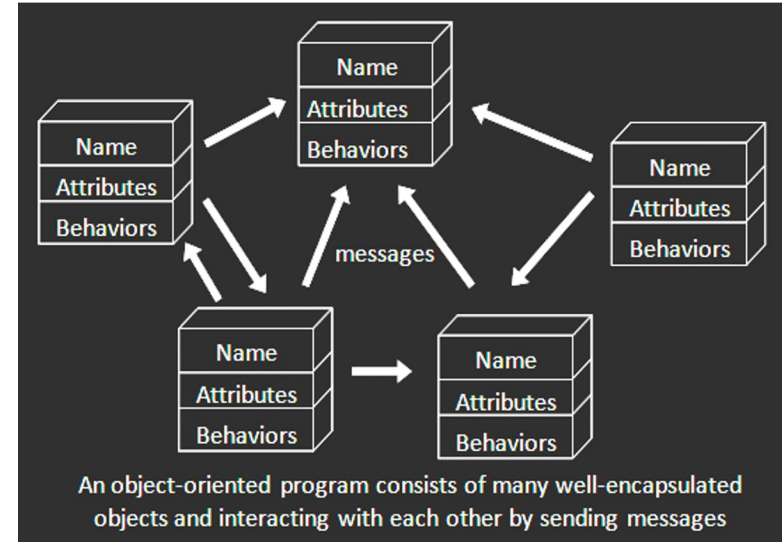 - With <u>methods</u> you ask of them
    - These are the behaviors
  - With <u>local state</u>, to remember
    - These are the attributes

- **<u>Classes</u> & <u>Instances</u>**
  - Instance an example of class
  - E.g., Fluffy is instance of Dog

- **<u>Inheritance</u>** saves code
  - Hierarchical classes
  - e.g., a Tesla is a special case of an Electric Vehicle, which is a special cade of a car

- Other Examples (though not pure)
  - Java (CS61B), C++

An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

www3.ntu.edu.sg/home/ehchua/programming /java/images/OOP-Objects.gif
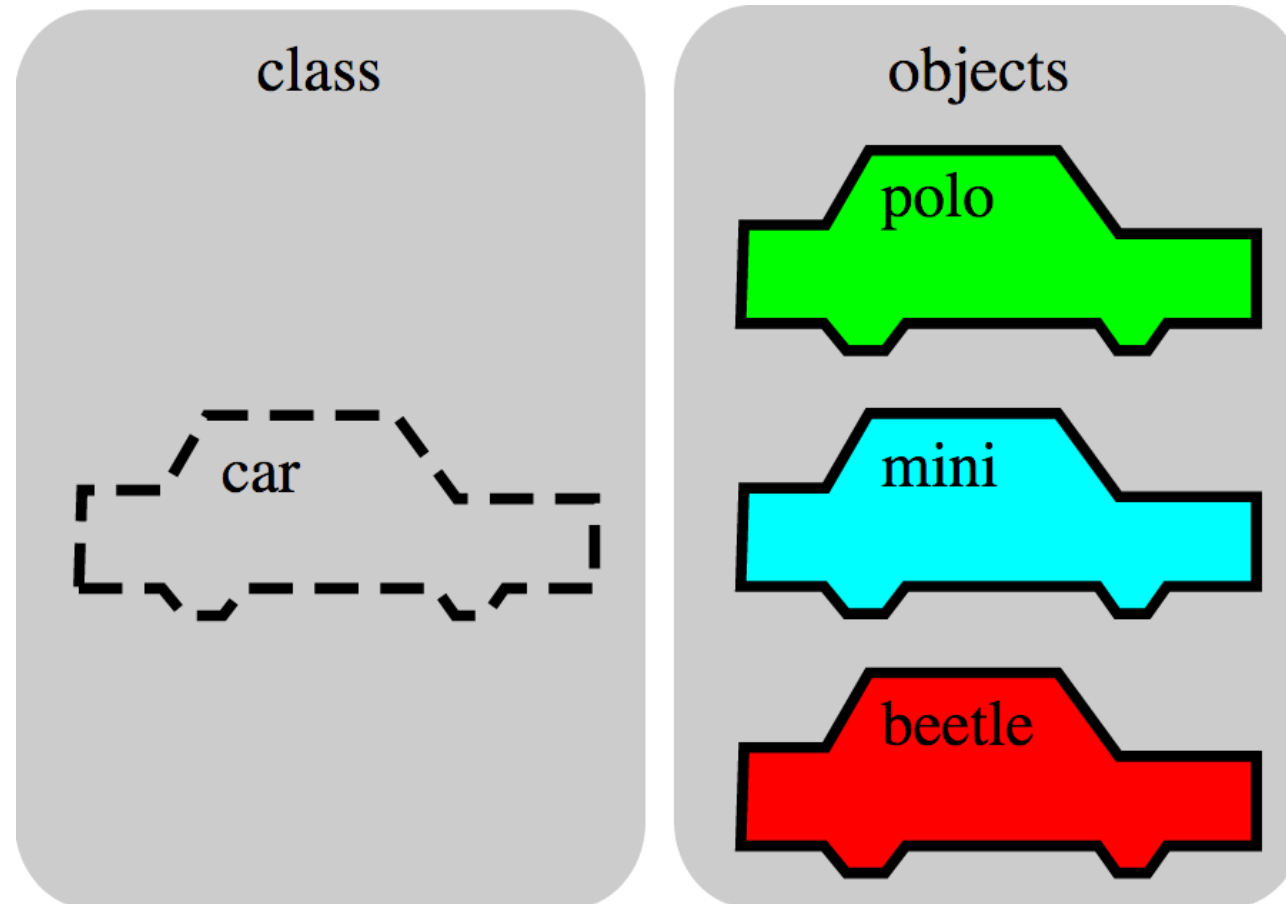
Classes

- Consist of data and behavior, bundled together to create abstractions

  - Abstract Data Types use functions to create abstractions

  - Classes define a new **type** in a programming language

    - They make the "abstract" data type concrete.

- A class has

- attributes (variables)

- methods (functions)

  that define its behavior.
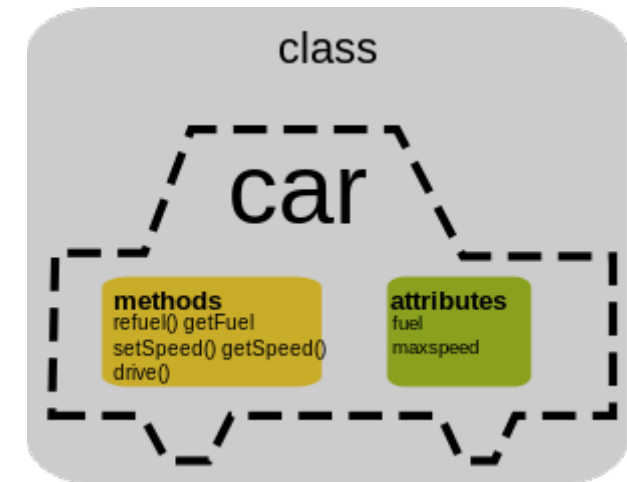
Objects

- An **object** is the instance of a class.

Objects

- Objects are concrete instances of classes in memory.
- They have *state*

  - mutable vs immutable (lists vs tuples)
- Methods are functions that belong to an object

  - 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:
    def __init__(self):
        <initialization steps>

    .
    .
    .
    <statement-N>



# Coming Next Week:
class ClassName ( inherits ):
    <statement-1>

    .
    .
    .
    <statement-N>
```

From ADTs to Classes

- An ADT is an *abstract* representation of a *type* of Data.

```python
def points(x, y) # our point ADT
    return { 'x': x, 'y': y}


class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def subtract(self, other):
        return Point(self.x - other.x, self.y - other.y)
```

From ADTs to Classes (Usage)

```
>>> origin = point(0, 0)    # Using the ADT
>>> type(origin)
<class 'dict'>
>>> origin
{'x': 0, 'y': 0}
>>> my_house = Point(5, 5)   # Using the class
>>> my_house.x
5
>>> type(my_house)
<class '__main__.Point'>
>>> my_house
<__main__.Point object at 0x104fdc710>
```

```
class BaseAccount:

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

    def account_name(self):

        return self.name

    def 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

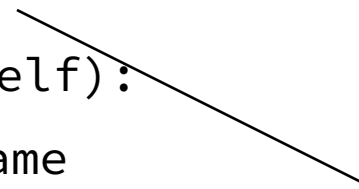# Special Initialization Method

```python
class BaseAccount:

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

    def account_name(self):

        return self.name

    def 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

```python
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: Suggested "private" attributes

```python
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

```python
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

```python
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())
```