UC Berkeley
EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Object-Oriented Programming

1

Announcements

- Midterm Regrades: Open Until 11:59pm Sunday Night
  - Please check to make sure everything looks good
- Maps Checkpoint is now due Friday Night
- Maps due 10/30, but HW 7 will be due Sunday 11/1
  - But we can't support weekend OH
- Just as a reminder: You have slip days for HW, Projects and Labs

2

Computing In The News: Election Related

- https://www.govtech.com/security/Experts-Florida-Votine-Machines-Ripe-for-Foreign-Hackers.html
  - Experts think Florida voting machines may be hackable...
  - [50 Florida Jokes Removed]
- https://www.nbcnews.com/news/us-news/california-may-replace-cash-bail-algorithms-some-worry-will-be-n1243750
  - Prop 15 in California: "California may replace cash bail with algorithms — but some worry that will be less fair"
  - CA Residents: Read up: It's confusing!

3

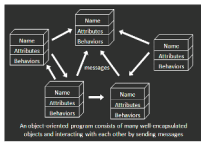
Learning Objectives

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

4

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++



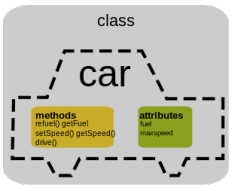
www3.ntu.edu.sg/home/ehchua/programming/java/images/OOP-Objects.gif

5

Classes

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

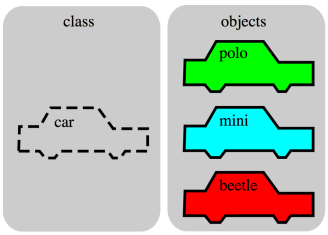
  that define its behavior.



6

Objects

• An object is the instance of a class.



7

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

8

Python class statement

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


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


9

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

10

Creating an object, invoking a method

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

The Class Constructor

dot

11

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

12

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

13

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
```


14

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
```


15

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
```


16

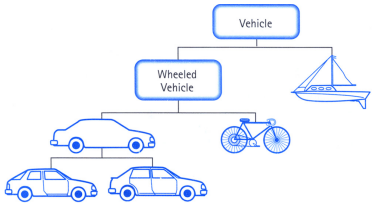
More class attributes

```
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())
```


17

Class Inheritance

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



18

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

19

Example

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


21

More special methods

```
class Account(BaseAccount):
    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)
```

Goal: unambiguous

Goal: readable

22

Classes using classes

```
class Bank:
    accounts = []

    def add_account(self, name, account_type,
                    initial_deposit):
        assert (account_type == 'savings') or
               (account_type == 'checking'), "Bad Account type"
        assert initial_deposit > 0, "Bad deposit"
        new_account = Account(name, account_type,
                              initial_deposit)
        Bank.accounts.append(new_account)

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


23