Computational Structures in Data Science

Object-Oriented Programming
Announcements

• Midterm Regrades: Open Until 11:59pm Friday Night
  - Please check to make sure everything looks good
  - If you see something wrong, click on the question, then at the bottom of the page select “Request Regrade”. If your solution is correct and we missed it, please explain why or show an example.

• Project 2, Ants vs Some Bees will be out soon! (Later this week or next.)
  - You’ll have about 3 weeks and can work with 1 other person.
Computing In The News: Election Related

• Researchers design an AI-powered backpack for the visually impaired
• Artificial intelligence researchers hope the bag will one day replace canes and guide dogs

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
  - With **local state**, to remember
- **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++
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
Creating an object, invoking a method

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

• Attributes of an object accessible with ‘dot’ notation
  \texttt{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”
  – \_\texttt{foo} and \_\_\texttt{foo} do different things inside a class.
    – \texttt{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
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())
Class Inheritance

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

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

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