Data Foundations: The Web and Object Oriented Programming

Instructor: Anthony Rios

Outline

Review

Questions

Questions? Want to revisit an exercise? Homework questions?

Remember Homework is **due Wednesday** by 12:59pm!

Quiz on **Wednesday**!

Review

Introduction

https://www.youtube.com/watch?v=I6nu8N6FQqc

File IO on the Web

example.py

```
import urllib.request

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
        print(line.decode().strip())
```

anthony@MacBook:~$ python example.py

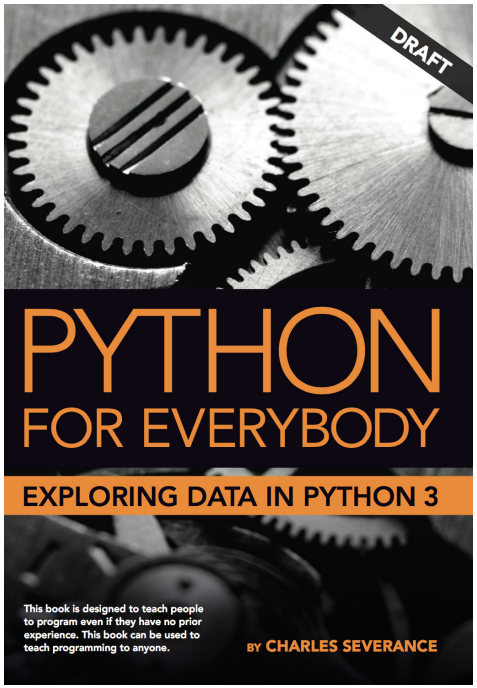
```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Reading binary files using urllib

example.py

```
import urllib.request

img = urllib.request.urlopen('http://data.pr4e.org/cover3.jpg').read()
fhand = open('cover3.jpg', 'wb')
fhand.write(img)
fhand.close()
```



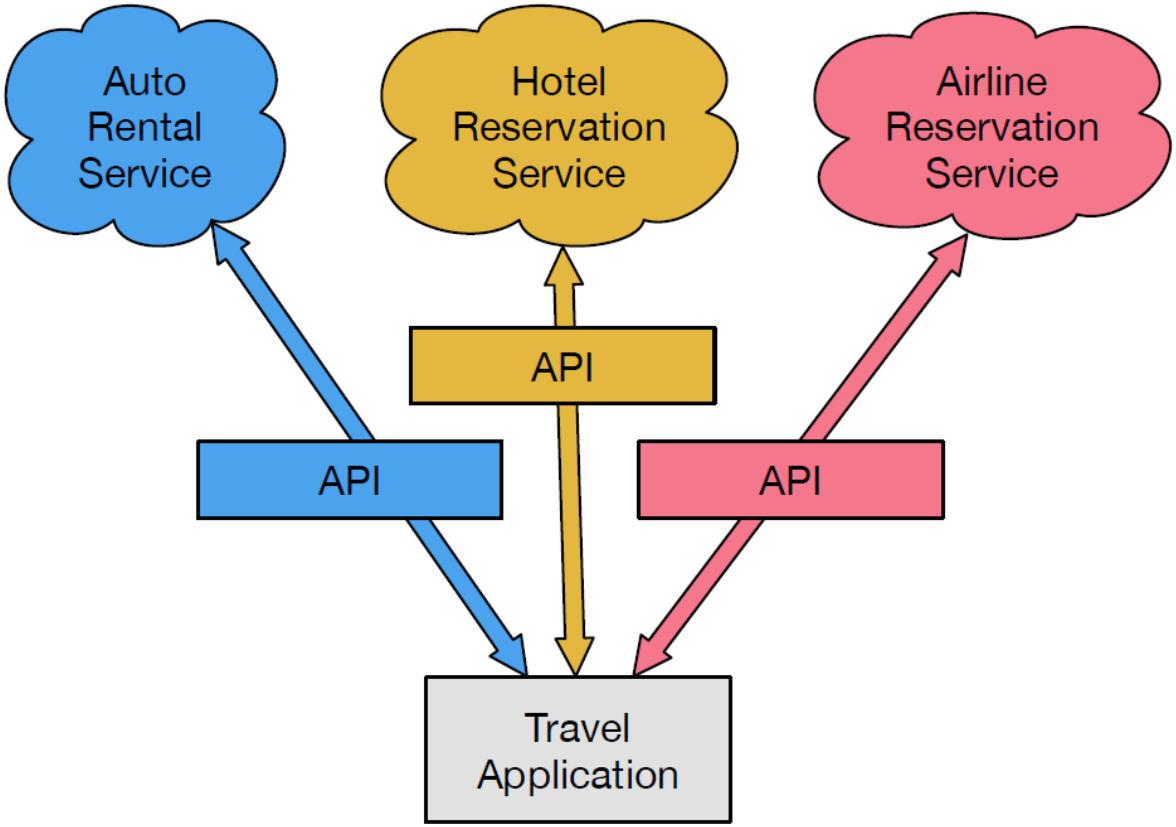
Exercise 3

Write a program that downloads and counts the total number of words in 'http://data.pr4e.org/romeo.txt'.


10 minutes

Rest API

https://www.youtube.com/watch?v=pHFWGN-upGM



API

When an application makes a set of services in its API available over the web, we call these **web services**.

Advantages of service architectures:

- We always maintain only **one copy** of the data.
- The **owners** of the data can set rules about the use of their data.

API: Geocoding Case Study

```
>>> import urllib.request, urllib.parse
>>> serviceurl = 'http://py4e-data.dr-chuck.net/json?'
>>> api_key = 42
>>> address = "Ann Arbor, MI"
>>> parms = {'address':address}
>>> parms['key'] = api_key
>>> url = serviceurl + urllib.parse.urlencode(parms)
>>> uh = urllib.request.urlopen(url)
>>> data = uh.read().decode()
>>> print(data) # Returns JSON
{"results" : [{ "address_components" : [{
              "long_name" : "700", ...
```

API: Web Forms

The previous code – with the exception of the api_key – can be used to submit web forms.

```
>>> serviceurl = 'http://duckduckgo.com/html/?'

>>> query = "Python"

>>> parms = {'q':query}

>>> url = serviceurl + urllib.parse.urlencode(parms)

>>> html_page = url.read().decode()
```

`http://duckduckgo.com/html/`

Exercise 4

Write code to query http://duckduckgo.com/html/ with the key query "data science". Parse the resulting page by returning all the unique web URLs. Return only the base URLs (http://duckduckgo.com, www.duckduckgo.com, ...)

hint: Use re.findall()


15 minutes

BeautifulSoup: Parsing Wikipedia Tables

```
>>> from bs4 import BeautifulSoup

>>> import urllib

>>> url = 'https://en.wikipedia.org/wiki/List_of_Asian_countries_by_area'

>>> html = urllib.request.urlopen(url).read()

>>> soup = BeautifulSoup(html,'lxml')

>>> print(soup.prettify())
```

Parsing Wikipedia Tables

cat example.py

```
from bs4 import BeautifulSoup
import urllib
url = 'https://en.wikipedia.org/wiki/List_of_Asian_countries_by_area'
html = urllib.request.urlopen(url).read()
soup = BeautifulSoup(html,'lxml')
myTable = soup.find('table',{'class':'wikitable sortable'})
tbody = myTable.find('tbody')
rows = tbody.find_all('tr')
for row in rows:
        cols = row.find_all('td')
        cols = [ele.text.strip() for ele in cols]
        print(cols)
```

Parsing Wikipedia Tables

python example.py

```
['18', 'Japan', '377,930', '']
['19', 'Vietnam', '331,212', '']
['20', 'Malaysia', '330,803', '']
['21', 'Oman', '309,500', '']
...
```

Exercise 3

Find another table on Wikipedia and repeat the process in the previous slide to parse that table.

Programming Patterns

We have discussed the 4 main programming patterns:

- Sequential Code
- Conditional Code (if, elif, else)
- Repetitive Code (loops)
- Store and Reuse (functions)

We also discussed basic data structures, like lists, sets and dictionaries.

Programming

Every programming tasks involves the use of **data structures** and writing **code that manipulates them**.

Elegance

There is always **multiple** ways to write a program.

However, some programs are “more elegant”, while other programs are “less elegant”

Object Oriented Programming

Object oriented programming is a way to arrange your code so that you can **zoom into 50 lines** of the code and **understand it** while **ignoring the other 999,950 lines** of code for the moment.

Object Oriented Programming

We have been using objects throughout this course!

```
>>> myList = []
```
is equivalent to...

```
>>> myList = list()
```

Object Oriented Programming

```
# This line CONSTRUCTS an object of type "list"
>>> stuff = list()

# The following lines CALL different METHODS
>>> stuff.append("python")

>>> stuff.append("chuck")

>>> stuff.sort()

>>> print(stuff[0])

>>> print(stuff.__getitem__(0))

>>> print(list.__getitem__(stuff, 0))
```

Methods vs Functions

A method is a **function** that is contained **within a class** and the objects that are constructed from the class.

An **object** contains both data and **methods** that manipulate that data.

- An object is active, not passive; it does things
- An object is responsible for its own data
  - But: it can expose that data to other objects

An object has state

An object contains both **data** and methods that manipulate that data

- The data represent the **state** of the object
- Data can also describe the relationships between this object and other objects
- Data is also known as "attributes"

Example: A "rabbit" object

Assume we want to create an object that represents a rabbit.

It would have **data**:

- How hungry it is
- How frightened it is
- Where it is

And **methods**:

- eat, hide, run, dig

Creating Objects in Python

example.py

```
class PartyAnimal:
        x = 0 # data/attribute
        def party(self): # Method
                self.x += 1
                print("So far", self.x)
```

Creating Objects in Python

example.py

```
class PartyAnimal:
        x = 0 # data/attribute
        def party(self): # Method
                self.x += 1
                print("So far", self.x) an = PartyAnimal()
an.party()
an.party()
an.party()
```

anthony@MacBook:~$ python example.py

```
So far 1
So far 2
So far 3
```

Classes as types

DEMO

- type() – Returns the type of an object

- dir() – Returns the methods in an object

Object life-cycle

example.py

```
class PartyAnimal:
        x = 0
        def __init__(self): # Called when object is created/initialized
                print('I am constructed')
        def party(self):
                self.x = self.x + 1
                print('So far',self.x)
        def __del__(self): # Called when object is destroyed
                print('I am destructed', self.x)
an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)
```

Object life-cycle

example.py

```
...
an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)
```

anthony@MacBook:~$ python example.py

```
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

Exercise 4

Write a class named "CheckingAccount" that contains the current **balance** of the account (an int) and the following methods:

- **__init__** – takes a "balance" parameter to initialize the data (balance) of the object.
- **withdraw** – takes an input parameter "amount" and modifies the data by reducing the balance. If "amount" results in an overdraw, subtract an extra 20 dollars. This method should return the balance balance.
- **deposit** – takes an input parameter "amount" and modifies the data by increasing the balance by "amount".

Write a few test cases to check your work.

Multiple Instances: Error on page 181 py4E

example.py

```
class PartyAnimal:
        x = 0
        name = ''
        def __init__(self, nam):
                self.name = nam
                print(self.name,'constructed')
        def party(self):
                self.x = self.x + 1
                print(self.name,'party count',self.x)
s = PartyAnimal('Sally')
j = PartyAnimal('Jim')
s.party()
j.party()
s.party()
```

anthony@MacBook:~$ python example.py

```
Sally constructed
Jim constructed
Sally party count 1
Jim party count 1
Sally party count 2
```

Inheritance

example.py

```
from party import PartyAnimal
class CricketFan(PartyAnimal):
        points = 0
        def six(self):
                self.points = self.points + 6
                self.party()
                print(self.name,"points",self.points)
s = PartyAnimal("Sally")
s.party()
j = CricketFan("Jim")
j.party()
j.six()
print(dir(j))
```

Inheritance

example.py

```
s = PartyAnimal("Sally")
s.party()
j = CricketFan("Jim")
j.party()
j.six()
```

anthony@MacBook:~$ python example.py

```
Sally constructed
Sally party count 1
Jim constructed
Jim party count 1
Jim party count 2
Jim points 6
```

Inheritance

Here we **override** the instructor of the original method __init__.

example.py

```
from party import PartyAnimal
class CricketFan(PartyAnimal):
        points = 0
        def __init__(self):
                PartyAnimal.__init__(self, "Anthony")
        def six(self):
                self.points = self.points + 6
                self.party()
                print(self.name,"points",self.points)
j = CricketFan() # No parameter needed now
```

anthony@MacBook:~$ python example.py

```
Anthony constructed
```

Exercise 5

Given the code for exercise 5, answer the following questions:

- What are the parent and child classes here?
- What does the code print out? (Try figuring it out without running it in Python)
- Which get description method is called when 'study spell(Confundo())' is executed? Why?
- What do we need to do so that 'print Accio()' will print the appropriate description ('This charm summons an object to the caster, potentially over a significant distance')? Change to code to do this.

Try to answer the questions before running the code!

Important Information

Homework 1 is due on Wednesday!

Quiz on **Wednesday**!