Web Services

First Lesson. (Most Important)

Data on the Web

- With the HTTP Request/Response well understood and well supported, there was a natural move toward exchanging data between programs using these protocols
- We needed to come up with an agreed way to represent data going between applications and across networks
- There are two commonly used formats: XML and JSON

Sending Data Across the "Net"

PHP Array

Python Dictionary

JavaScript Object

Java HashMap

a.k.a. "Wire Protocol" - What we send on the "wire"

Agreeing on a “Wire Format”

Python Dictionary

Serialize

```
<person>
  <name>
    Chuck
  </name>
  <phone>
    303 4456
  </phone>
</person>
```

De-Serialize

Java HashMap

XML

Agreeing on a “Wire Format”

Python Dictionary

Serialize

```
{
  "name" :  "Chuck",
  "phone" :  "303-4456"
}
```

De-Serialize

Java HashMap

JSON

XML

Marking up data to send across the network...

http://en.wikipedia.org/wiki/XML

XML “Elements” (or Nodes)

- Simple Element
- Complex Element

```
<people>
  <person>
    <name>Chuck</name>
    <phone>303 4456</phone>
  </person>
  <person>
    <name>Noah</name>
    <phone>622 7421</phone>
  </person>
</people>
```

eXtensible Markup Language

- Primary purpose is to help information systems share structured data
- It started as a simplified subset of the Standard Generalized Markup Language (SGML), and is designed to be relatively human-legible

http://en.wikipedia.org/wiki/XML

XML Basics

- Start Tag
- End Tag
- Text Content
- Attribute
- Self Closing Tag

```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

White Space

```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

Line ends do not matter.
White space is generally discarded on text elements.
We indent only to be readable.

```
<person>
  <name>Chuck</name>
  <phone type="intl">+1 734 303 4456</phone>
  <email hide="yes" />
</person>
```

XML Terminology

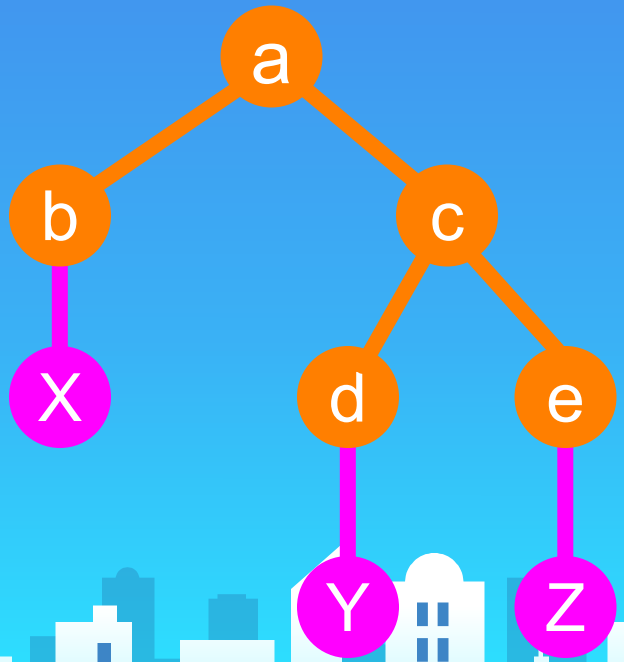
- Tags indicate the beginning and ending of elements
- Attributes - Keyword/value pairs on the opening tag of XML
- Serialize / De-Serialize - Convert data in one program into a common format that can be stored and/or transmitted between systems in a programming language-independent manner

http://en.wikipedia.org/wiki/Serialization

XML as a Tree

```
<a>
  <b>X</b>
  <c>
    <d>Y</d>
    <e>Z</e>
  </c>
</a>
```

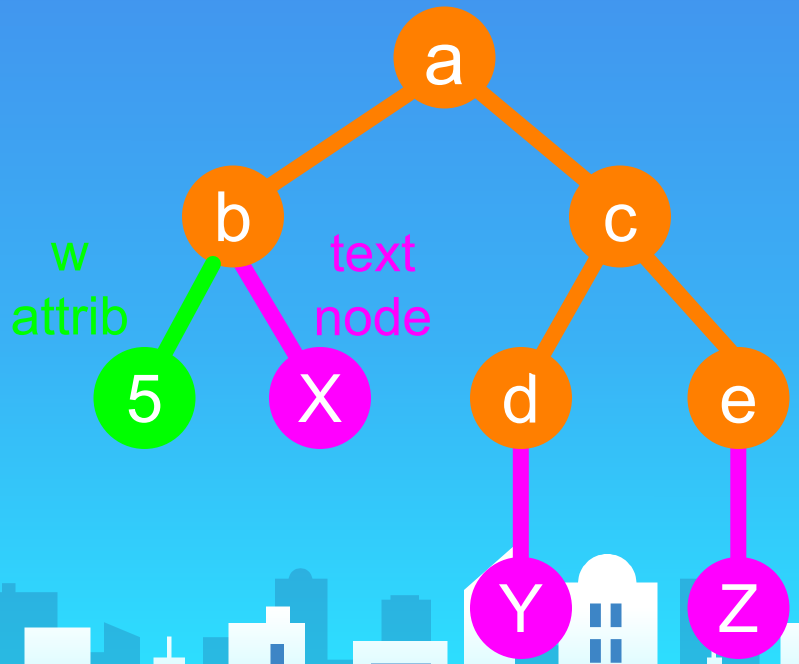
Elements Text



XML Text and Attributes

```
<a>
  <b w="5">X</b>
  <c>
    <d>Y</d>
    <e>Z</e>
  </c>
</a>
```

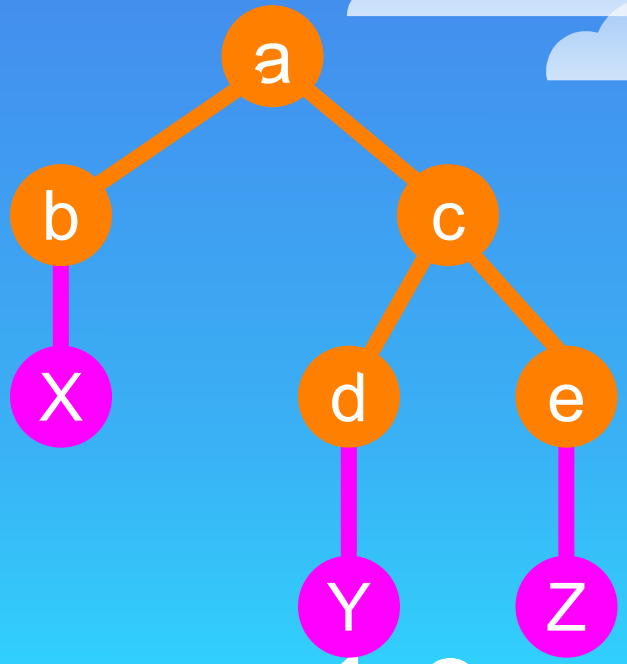
Elements Text



XML as Paths

```
<a>
  <b>X</b>
  <c>
    <d>Y</d>
    <e>Z</e>
  </c>
</a>
```

/a/b	X
/a/c/d	Y
/a/c/e	Z



Elements Text

xml1.py

```
import xml.etree.ElementTree as ET
data = '''<person>
  <name>Chuck</name>
  <phone type="intl">
     +1 734 303 4456
   </phone>
   <email hide="yes"/>
</person>'''

tree = ET.fromstring(data)
print('Name:',tree.find('name').text)
print('Attr:',tree.find('email').get('hide'))
```

xml2.py

```
import xml.etree.ElementTree as ET
input = '''<stuff>
    <users>
        <user x="2">
            <id>001</id>
            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>
            <name>Brent</name>
        </user>
    </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))
for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get("x"))
```

JavaScript Object Notation

JavaScript Object Notation

- Douglas Crockford - “Discovered” JSON
- Object literal notation in JavaScript

http://www.youtube.com/watch?v=kc8BAR7SHJI

JSON

http://www.json.org/

Introducing JSON

العربية Български 中文 Český Nederlandse Dansk English Esperanto Française Deutsch Ελληνικά עברית Magyar Indonesia
Italiano 日本 한국어 فارسی Polski Português Română Русский Српски Slovenščina Español Svenska Türkçe Tiếng Việt

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right

object
{}
{ *members* }
members
pair
pair , *members*
pair
string : *value*
array
[]
[*elements*]
elements
value
value , *elements*
value
string
number
object

```
import json
data = '''{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
   },
   "email" : {
     "hide" : "yes"
   }
}'''

info = json.loads(data)
print('Name:',info["name"])
print('Hide:',info["email"]["hide"])
```

json1.py

JSON represents data as nested “lists” and “dictionaries”

```
import json
input = '''[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  } ,
  { "id" : "009",
    "x" : "7",
    "name" : "Chuck"
  }
]'''

info = json.loads(input)
print('User count:', len(info))
for item in info:
    print('Name', item['name'])
    print('Id', item['id'])
    print('Attribute', item['x'])
```

json2.py

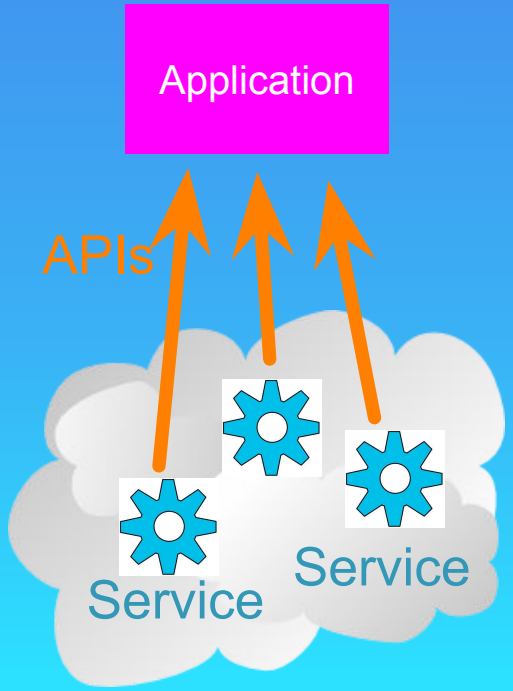
JSON represents data as nested “lists” and “dictionaries”

Service Oriented Approach

http://en.wikipedia.org/wiki/Service-oriented_architecture

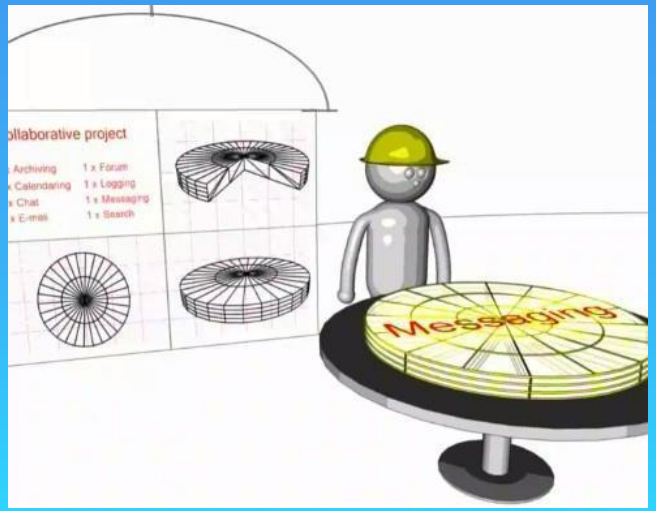
Service Oriented Approach

- Most non-trivial web applications use services
- They use services from other applications
  - Credit Card Charge
  - Hotel Reservation systems
- Services publish the “rules” applications must follow to make use of the service (API)



Multiple Systems

- Initially - two systems cooperate and split the problem
- As the data/service becomes useful - multiple applications want to use the information / application



http://www.youtube.com/watch?v=mj-kCFzF0ME

5:15

Web Services

http://en.wikipedia.org/wiki/Web_services

Application Program Interface

The API itself is largely abstract in that it specifies an interface and controls the behavior of the objects specified in that interface. The software that provides the functionality described by an API is said to be an “implementation” of the API. An API is typically defined in terms of the programming language used to build an application.

http://en.wikipedia.org/wiki/API

API Call

```
import urllib.request, urllib.parse, urllib.error

import json

serviceurl =
'https://api.rss2json.com/v1/api.json?rss_url=https%3A%2F%2Ftechcrunch.com%2Ffeed%2F'

uh = urllib.request.urlopen(serviceurl)

data = uh.read().decode()

js = json.loads(data)

if js['status'] != 'ok':

    print('Fail to retrive data ...')

for item in js['items']:

    print(item['title'])
```

Call to http://bit.ly/jsonpractice and parse title

Assignment

Call to http://bit.ly/jsonpractice and parse description

Assignment

Call to http://bit.ly/jsonpractice and parse url

Assignment

Modules

Second Lesson

What is Module?

- Consider a module to be the same as a code library.
- A file containing a set of functions you want to include in your application.

Mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

Mymodule.py

```
import mymodule

mymodule.greeting("Jonathan")
```

Mymodule.py

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Import custom Module

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Renaming Module

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Built In Modules

```
import platform

x = platform.system()
print(x)
```

Dir Functions

```
import platform

x = dir(platform)
print(x)
```

Import From Module

```
from mymodule import person1

print (person1["age"])
```

Create your own module for accounts

Assignment

Create your own module to list available files.

Assignment