A Note About `try/except`

- `try` should surround only a brief snippet of code that you actually believe might fail, and you know why
  - For example, opening a filename provided by the user, which may not exist
  - Expecting a user to input a number, but not being sure they followed instructions
  - Do not use it to around a large portion of code just in case something goes wrong. That typically means you don’t understand what the code is doing!
Web Services, XML, APIs
Web Services
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”

Python Dictionary → “Wire Protocol” → Java HashMap

a.k.a. “Wire Protocol” – What we send on the “wire”
Agreeing on a “Wire Format”

Python Dictionary

Serialize

De-Serialize

Java HashMap

XML

<person>
  <name>Chuck</name>
  <phone>303 4456</phone>
</person>
<person>
  <name> Chuck </name>
  <phone> 303-4456 </phone>
</person>
Agreeing on a “Wire Format”

Python Dictionary → Serialize

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

De-Serialize → Java HashMap

JSON
XML vs. JSON

**XML:**

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

**JSON:**

```json
{"name": "Chuck", "phone": "303-4456"}
```
XML “Elements” (or Nodes)

- **Simple element:**
  ```xml
  <person>
    <name> Chuck </name>
    <phone> 303-4456 </phone>
  </person>
  ```

- **Complex element:**
  ```xml
  <people>
    <person>
      <name> Chuck </name>
      <phone> 303-4456 </phone>
    </person>
    
    <person>
      <name> Noah </name>
      <phone> 622-7421 </phone>
    </person>
  </people>
  ```
XML:
eXtensible Markup Language

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

XML Basics

• Start Tag

• End Tag

• Text Content

• Attribute

• Self Closing Tag

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

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>

<person>
  <name>Chuck</name>
  <phone type="intl">+1 734 303 4456</phone>
  <email hide="yes" />
</person>
Can View That as a Tree

```xml
<person>
  <name> Chuck </name>
  <phone type="intl"> +1 734 303-4456 </phone>
  <email hide="yes"/>
</person>
```
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">
  <title>Basic bread</title>
  <ingredient amount="8" unit="dL">Flour</ingredient>
  <ingredient amount="10" unit="grams">Yeast</ingredient>
  <ingredient amount="4" unit="dL" state="warm">Water</ingredient>
  <ingredient amount="1" unit="teaspoon">Salt</ingredient>
  <instructions>
    <step>Mix all ingredients together.</step>
    <step>Knead thoroughly.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Knead again.</step>
    <step>Place in a bread baking tin.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Bake in the oven at 180(degrees)C for 30 minutes.</step>
  </instructions>
</recipe>

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

XML as a Tree

```
<a>
  <b>X</b>
  <c>
    <d>Y</d>
    <e>Z</e>
  </c>
</a>
```
XML Text and Attributes

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

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

Elements  Text

/a/b  X
/a/c/d  Y
/a/c/e  Z
Parsing XML (See Code)

- `fromstring` converts the string representation of the XML into a “tree” of XML nodes.

- When the XML is in a tree, we have a series of methods we can call to extract portions of data from the XML.

- The `find` function searches through the XML tree and retrieves a node that matches the specified tag.

- Each node can have some text, some attributes (like hide), and some “child” nodes.

- Each node can be the top of a tree of nodes.
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')
```python
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')
```
XML Schema

Describing a “contract” as to what is acceptable XML.

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

- Description of the legal format of an XML document
- Expressed in terms of constraints on the structure and content of documents
- Often used to specify a “contract” between systems - “My system will only accept XML that conforms to this particular Schema.”
- If a particular piece of XML meets the specification of the Schema - it is said to “validate”

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

XML Schema

Contract

Validator

XML Validation
XML Document

<person>
  <lastname>Severance</lastname>
  <age>17</age>
  <dateborn>2001-04-17</dateborn>
</person>

XML Schema Contract

<xsd:complexType name="person">
  <xsd:sequence>
    <xsd:element name="lastname" type="xsd:string"/>
    <xsd:element name="age" type="xsd:integer"/>
    <xsd:element name="dateborn" type="xsd:date"/>
  </xsd:sequence>
</xsd:complexType>
Many XML Schema Languages

- Document Type Definition (DTD)
- Standard Generalized Markup Language (ISO 8879:1986 SGML)
- XML Schema from W3C - (XSD)
XSD XML Schema (W3C spec)

- We will focus on the World Wide Web Consortium (W3C) version
- It is often called “W3C Schema” because “Schema” is considered generic
- More commonly it is called XSD because the file names end in .xsd

http://www.w3.org/XML/Schema
XSD Structure

- **xs:element**

- **xs:sequence**

- **xs:complexType**

```xml
<xs:complexType name="person">
  <xs:sequence>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="age" type="xs:integer"/>
    <xs:element name="dateborn" type="xs:date"/>
  </xs:sequence>
</xs:complexType>
```

```xml
<person>
  <lastname>Severance</lastname>
  <age>17</age>
  <dateborn>2001-04-17</dateborn>
</person>
```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"
                   minOccurs="1" maxOccurs="1" />
      <xs:element name="child_name" type="xs:string"
                   minOccurs="0" maxOccurs="10" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Jim</child_name>
  <child_name>Borge</child_name>
</person>

http://www.w3schools.com/Schema/schema_complex_indicators.asp
<xs:element name="customer" type="xs:string"/>
<xs:element name="start" type="xs:date"/>
<xs:element name="startdate" type="xs:dateTime"/>
<xs:element name="prize" type="xs:decimal"/>
<xs:element name="weeks" type="xs:integer"/>

It is common to represent time in UTC/GMT, given that servers are often scattered around the world.

<customer>John Smith</customer>
<start>2002-09-24</start>
<startdate>2002-05-30T09:30:10Z</startdate>
<prize>999.50</prize>
<weeks>30</weeks>

http://www.w3schools.com/Schema/schema_dtypes_numeric.asp
ISO 8601 Date/Time Format

2002-05-30T09:30:10Z

- Year-month-day
- Time of day
- Timezone - typically specified in UTC / GMT rather than local time zone.

JavaScript Object Notation
Introducing JSON

ECMA-404 The JSON Data Interchange Standard.

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 brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).
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']

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']

JSON represents data as nested “lists” and “dictionaries”
JSON vs. XML

- Differences:
  - In XML, we can add attributes like “intl” to the “phone” tag. In JSON, we only have key-value pairs.
  - The XML “person” tag is gone, replaced by a set of outer curly braces.
  - JSON structures are simpler than XML because JSON has fewer capabilities than XML.
  - But JSON’s advantage: it maps directly to some combination of dictionaries and lists.
  - Since many languages have structures like lists and dictionaries, JSON is a very natural format to have two cooperating programs exchange data.
  - JSON is quickly becoming the format of choice for nearly all data exchange between applications because of its relative simplicity compared to XML.
APIs: Application Programming Interfaces
Service Oriented Approach

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

http://www.youtube.com/watch?v=mj-kCFzF0ME
Play up to about 2:40
A Service-Oriented Architecture
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
The Google Maps Geocoding API is a service that provides geocoding and reverse geocoding of addresses.

**Geocoding** is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude), which you can use to place markers on a map, or position the map.

This service is also available as part of the client-side Google Maps JavaScript API, or for server-side use with the Java Client, Python Client, Go Client and Node.js Client for Google Maps Services.
http://maps.googleapis.com/maps/api/geocode/json?
sensor=false&address=Ann+Arbor%2C+MI
import urllib
import json

serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'

while True:
    address = raw_input('Enter location: ')  
    if len(address) < 1 : break

    url = serviceurl + urllib.urlencode({'sensor':'false',
                                         'address': address})
    print 'Retrieving', url
    uh = urllib.urlopen(url)
    data = uh.read()
    print 'Retrieved',len(data),'characters'
    try: js = json.loads(str(data))
    except: js = None
    if 'status' not in js or js['status'] != 'OK':
        print '==== Failure To Retrieve ===='
        print data
        continue

    print json.dumps(js, indent=4)

    lat = js['results'][0]['geometry']['location']['lat']
    lng = js['results'][0]['geometry']['location']['lng']
    print 'lat',lat,'lng',lng
    location = js['results'][0]['formatted_address']
    print location

Enter location: Ann Arbor, MI
Retrieving http://maps.googleapis.com/...
Retrieved 1669 characters
lat 42.2808256 lng -83.7430378
Ann Arbor, MI, USA
Enter location:
API Security and Rate Limiting

- The compute resources to run these APIs are not “free”
- The data provided by these APIs is usually valuable
- The data providers might limit the number of requests per day, demand an API “key”, or even charge for usage
- They might change the rules as things progress...
Google Maps Geocoding API Usage Limits

The Google Maps Geocoding API has the following limits in place:

### Standard Usage Limits

Users of the standard API:
- 2,500 free requests per day, calculated as the sum of client-side and server-side queries.
- 50 requests per second, calculated as the sum of client-side and server-side queries.

Enable pay-as-you-go billing to unlock higher quotas:
- $0.50 USD / 1000 additional requests, up to 100,000 daily.

[ENABLE BILLING]

### Premium Usage Limits

Google Maps APIs Premium Plan customers:
- Shared daily free quota of 100,000 requests per 24 hours; additional requests applied against the annual purchase of Maps APIs Credits.
- 50* server-side requests per second.

* Default limit. Contact your Google Enterprise Sales Account Manager if you need a higher limit. Note that the client-side service offers Unlimited requests per second, per project.

Additional benefits of a Premium plan:
- Annual contracts with enterprise terms
- 24 hour technical support
- Service level agreement (SLA)
- Licenses for internal, OEM, and asset tracking use cases

[Contact Sales] for more info.
Authentication & Authorization

Twitter supports a few authentication methods and with a range of OAuth authentication styles you may be wondering which method you should be using. When choosing which authentication method to use you should understand the way that method will affect your users experience and the way you write your application.

Some of you may already know which type of authentication method you want to use and we want to help you check you've made the right choice.

<table>
<thead>
<tr>
<th>If you use the...</th>
<th>Send...</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST API</td>
<td>OAuth signed or application-only auth requests</td>
</tr>
<tr>
<td>Search API</td>
<td>OAuth signed or application-only auth requests</td>
</tr>
<tr>
<td>Streaming API</td>
<td>OAuth signed</td>
</tr>
</tbody>
</table>