

# Using Web Services

## Chapter 13

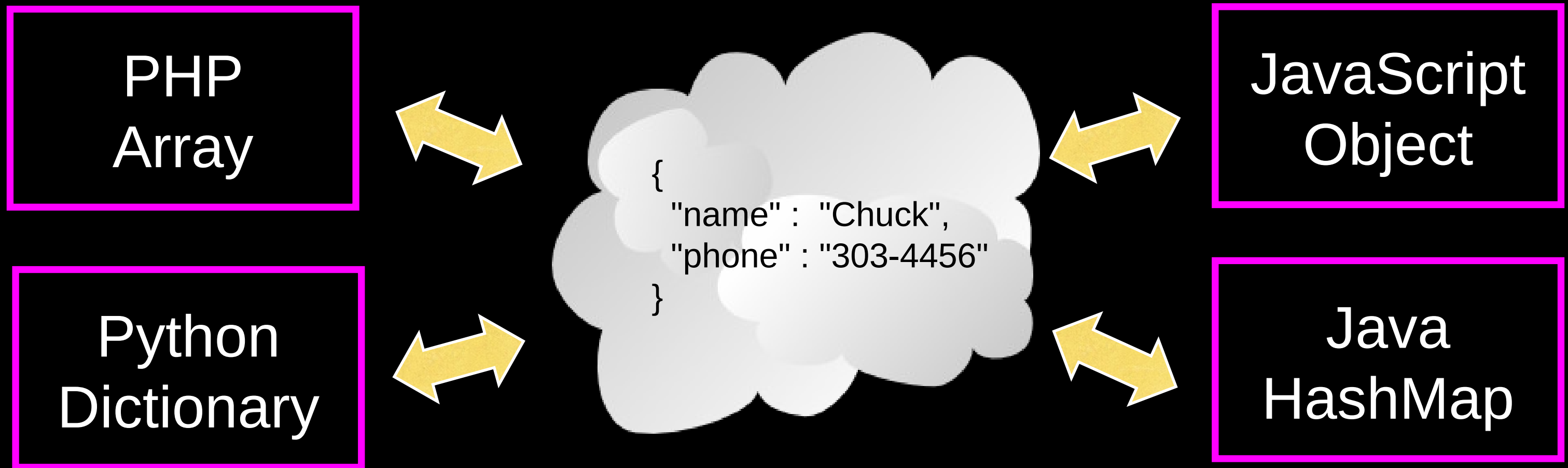
Python for Everybody  
[www.py4e.com](http://www.py4e.com)



# 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”



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

# Agreeing on a “Wire Format”

Python  
Dictionary

Serialize

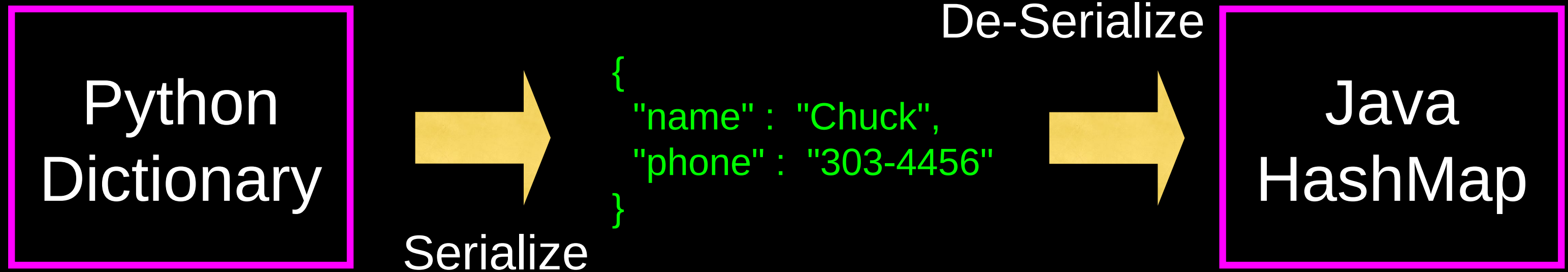
```
<person>  
  <name>  
    Chuck  
  </name>  
  <phone>  
    303 4456  
  </phone>  
</person>
```

De-Serialize

Java  
HashMap

XML

# Agreeing on a “Wire Format”



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

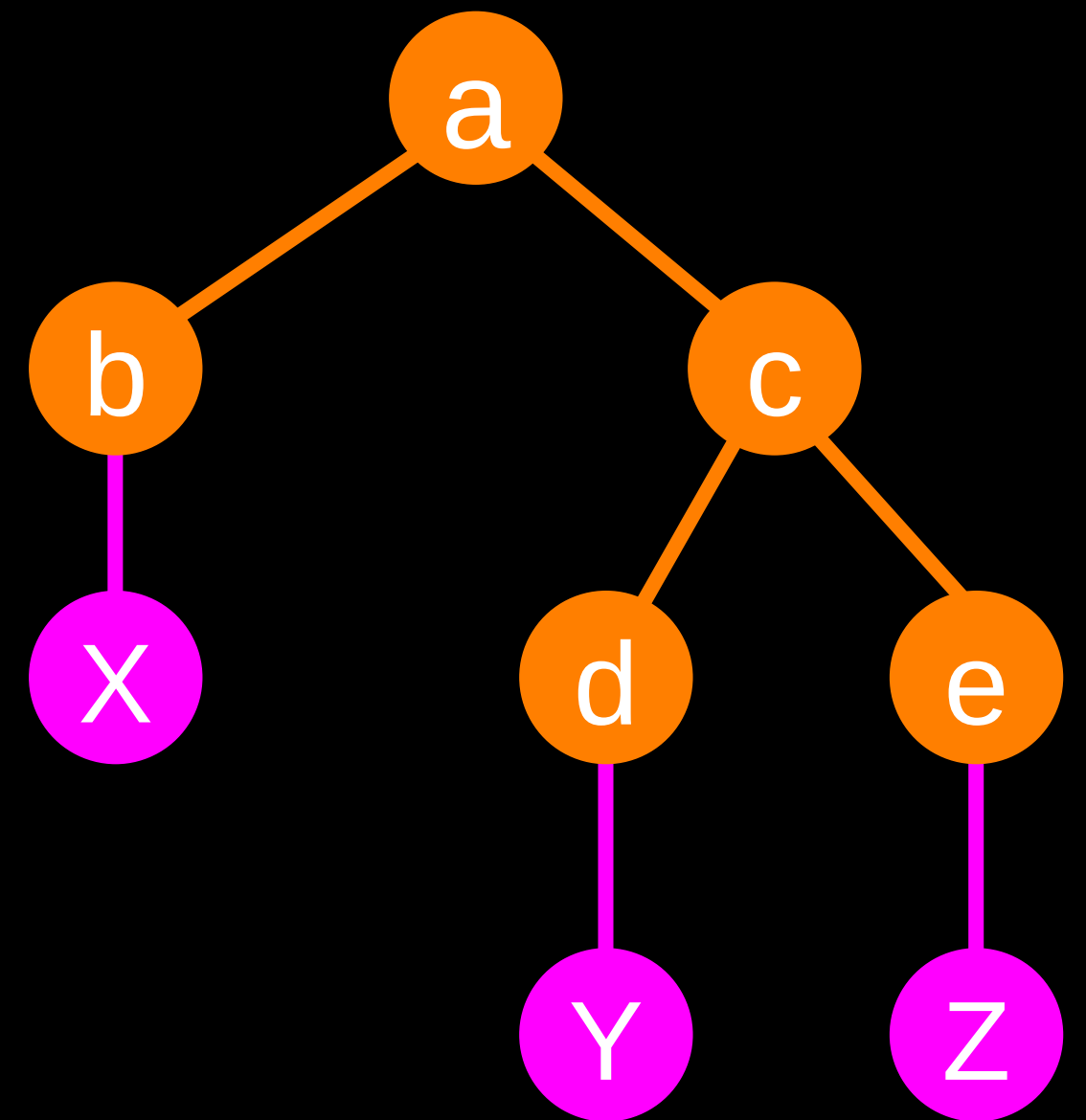
<http://en.wikipedia.org/wiki/Serialization>

# XML as a Tree

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

Elements

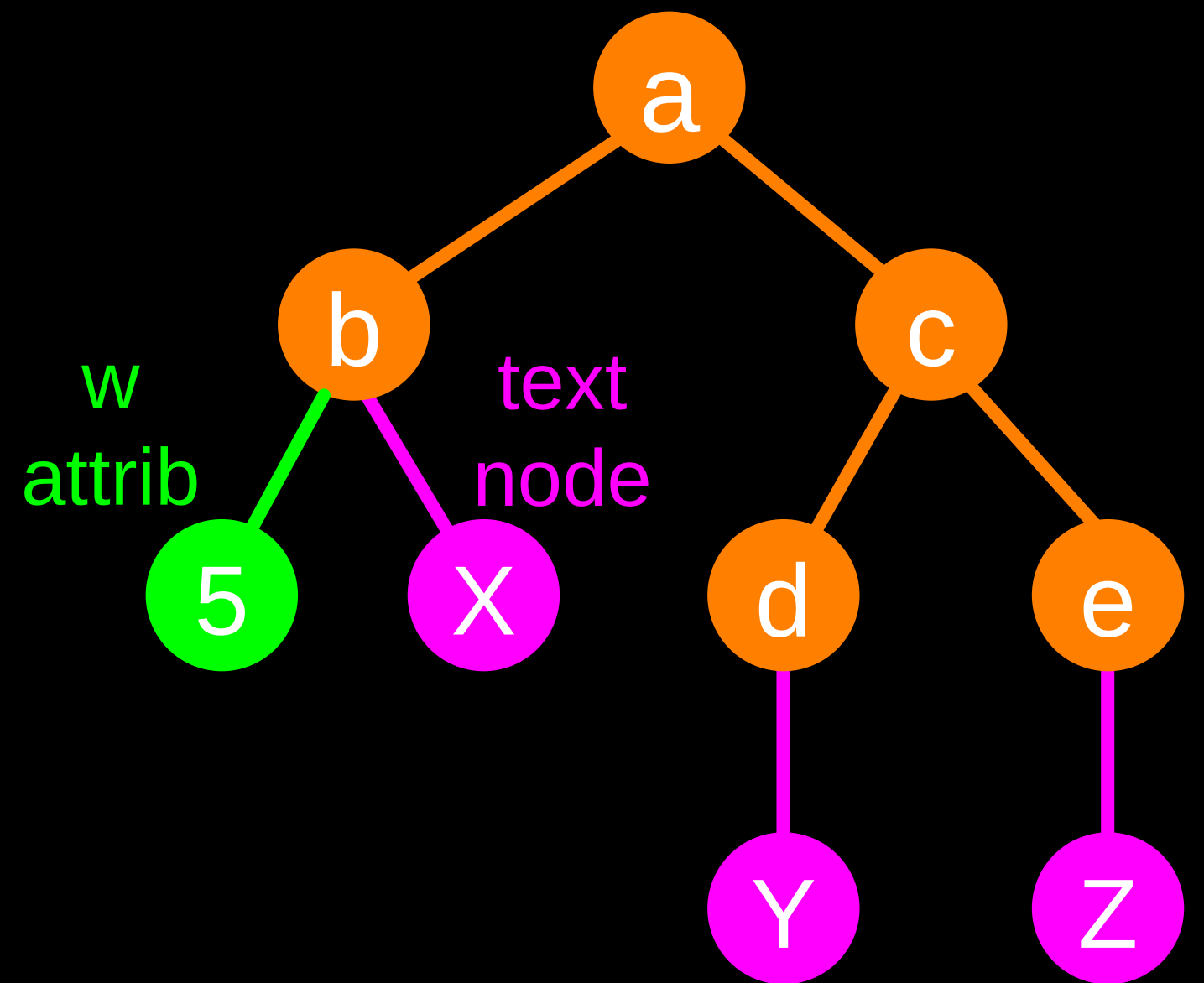
Text



# XML Text and Attributes

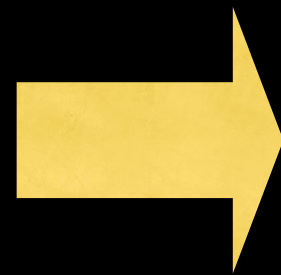
```
<a>  
  <b w="5">X</b>  
  <c>  
    <d>Y</d>  
    <e>Z</e>  
  </c>  
</a>
```

Elements      Text

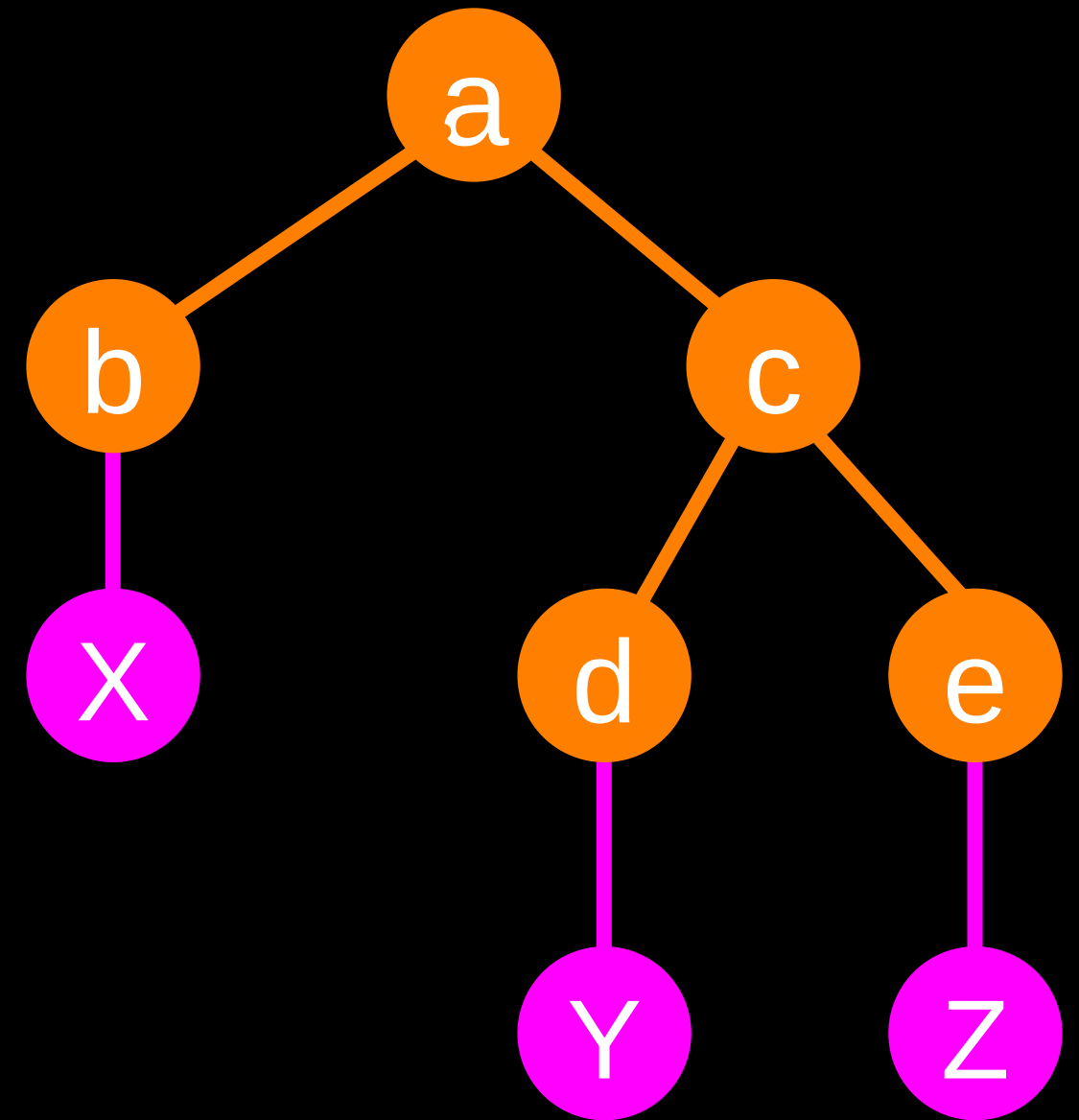


# XML as Paths

```
<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

# XML Schema

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

[http://en.wikipedia.org/wiki/Xml\\_schema](http://en.wikipedia.org/wiki/Xml_schema)

[http://en.wikibooks.org/wiki/XML\\_Schema](http://en.wikibooks.org/wiki/XML_Schema)

# XML Schema

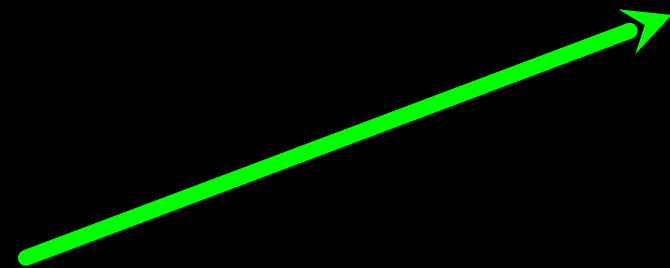
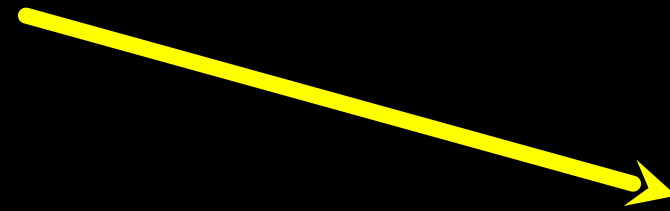
- Description of the **legal format** of an XML document
- Expressed in terms of constraints on the structure and content of documents
- Often used to specify a “**contract**” between systems - “My system will only accept XML that conforms to this particular Schema.”
- If a particular piece of XML meets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

[http://en.wikipedia.org/wiki/XML\\_schema](http://en.wikipedia.org/wiki/XML_schema)

# XML Validation

XML  
Document

XML Schema  
Contract



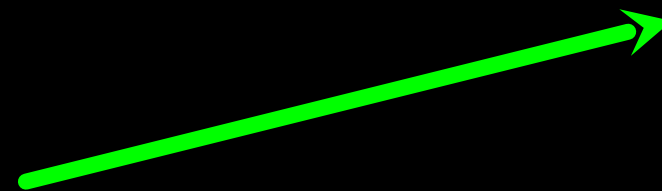
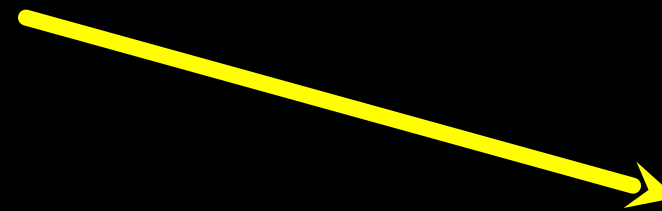
# XML Validation

## XML Document

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

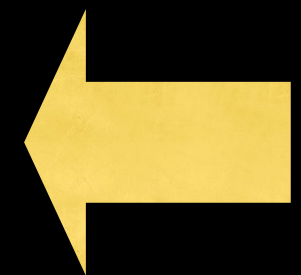
## XML Schema Contract

```
<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>
```



# Many XML Schema Languages

- Document Type Definition (DTD)
  - [http://en.wikipedia.org/wiki/Document\\_Type\\_Definition](http://en.wikipedia.org/wiki/Document_Type_Definition)
- Standard Generalized Markup Language (ISO 8879:1986 SGML)
  - <http://en.wikipedia.org/wiki/SGML>
- XML Schema from W3C - (XSD)
  - [http://en.wikipedia.org/wiki/XML\\_Schema\\_\(W3C\)](http://en.wikipedia.org/wiki/XML_Schema_(W3C))



[http://en.wikipedia.org/wiki/Xml\\_schema](http://en.wikipedia.org/wiki/Xml_schema)

# 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>

[http://en.wikipedia.org/wiki/XML\\_Schema\\_\(W3C\)](http://en.wikipedia.org/wiki/XML_Schema_(W3C))

# XSD Structure

- xs:element
- xs:sequence
- xs:complexType

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

```
<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>
```

# XSD

## Constraints

```
<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>
```

# XSD Data Types


```
<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>
```

# ISO 8601 Date/Time Format

2002-05-30T09:30:10Z

  
Year-month-day

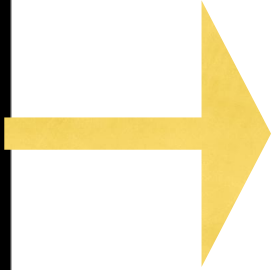
  
Time of  
day

  
Timezone - typically  
specified in UTC / GMT  
rather than local time  
zone

[http://en.wikipedia.org/wiki/ISO\\_8601](http://en.wikipedia.org/wiki/ISO_8601)

[http://en.wikipedia.org/wiki/Coordinated\\_Universal\\_Time](http://en.wikipedia.org/wiki/Coordinated_Universal_Time)

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Address">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Recipient" type="xs:string" />
        <xs:element name="House" type="xs:string" />
        <xs:element name="Street" type="xs:string" />
        <xs:element name="Town" type="xs:string" />
        <xs:element minOccurs="0" name="County" type="xs:string" />
        <xs:element name="PostCode" type="xs:string" />
        <xs:element name="Country">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="FR" />
              <xs:enumeration value="DE" />
              <xs:enumeration value="ES" />
              <xs:enumeration value="UK" />
              <xs:enumeration value="US" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```
<?xml version="1.0" encoding="utf-8" ?>
<Address
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="SimpleAddress.xsd">
  <Recipient>Mr. Walter C. Brown</Recipient>
  <House>49</House>
  <Street>Featherstone Street</Street>
  <Town>LONDON</Town>
  <PostCode>EC1Y 8SY</PostCode>
  <Country>UK</Country>
</Address>
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

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"))
```

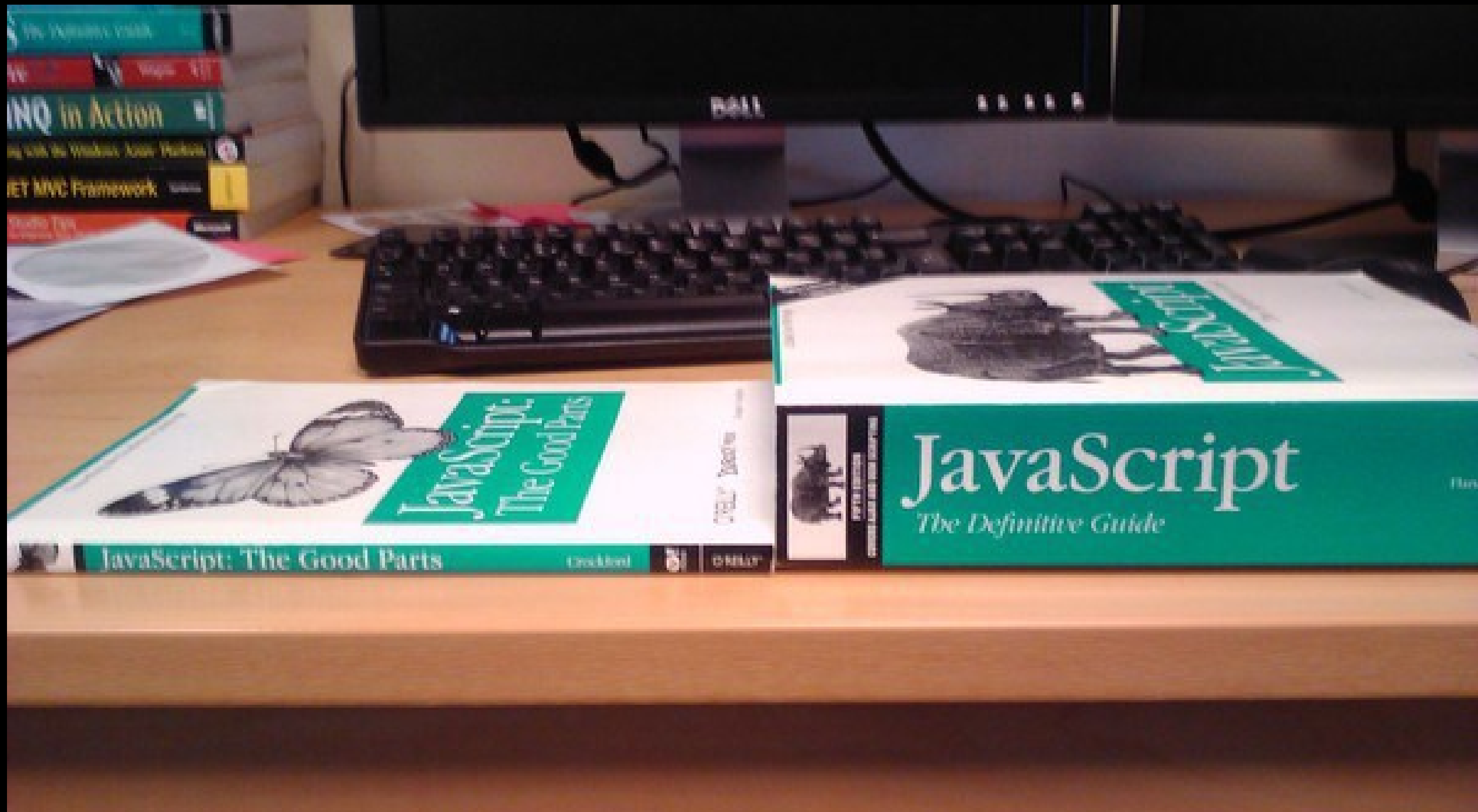
# JavaScript Object Notation

# JavaScript Object Notation

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




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



JSON

http://www.json.org/ Google



# 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
```

json1.py

```
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”

json2.py

```
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'])
```

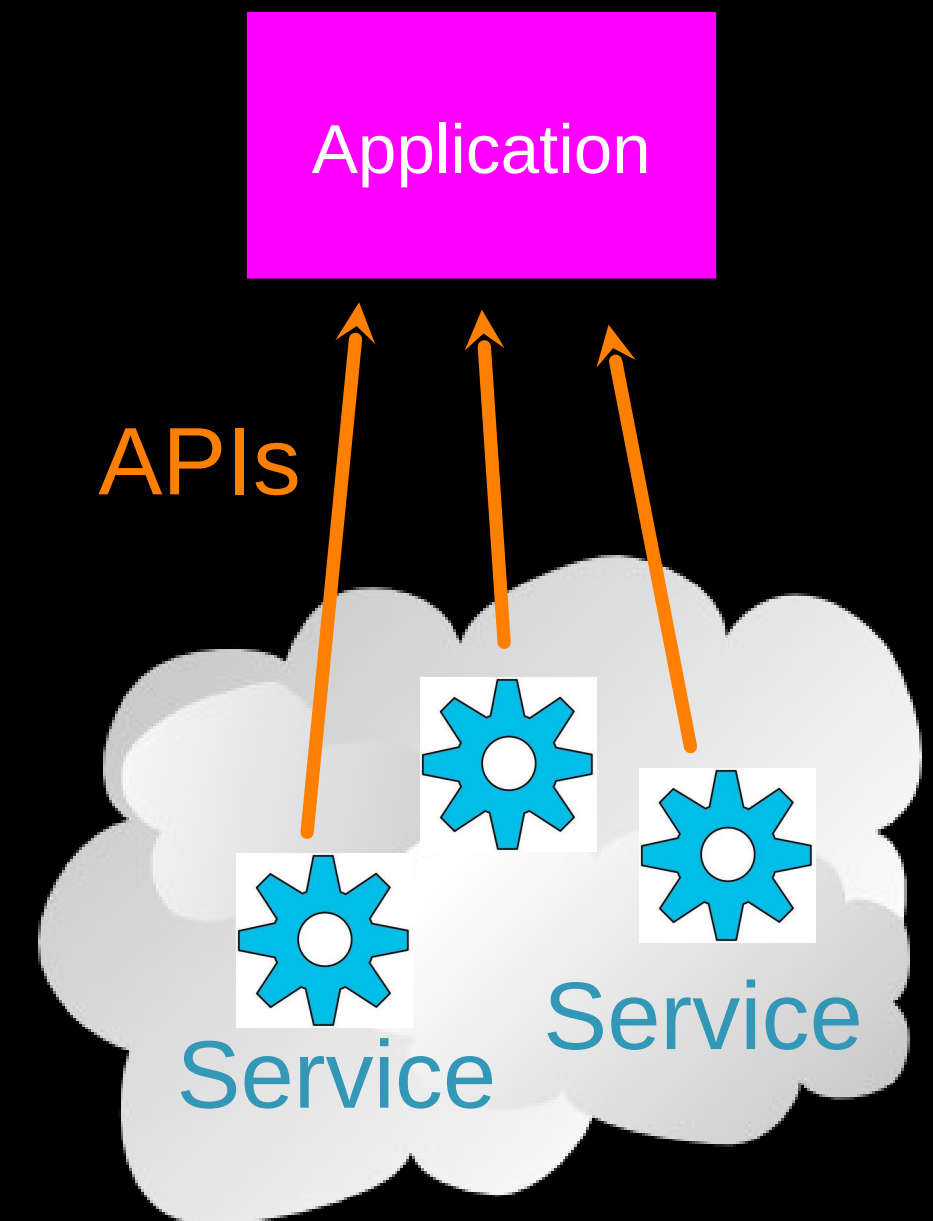
JSON represents data  
as nested “lists” and  
“dictionaries”

# Service Oriented Approach

[http://en.wikipedia.org/wiki/Service-oriented\\_architecture](http://en.wikipedia.org/wiki/Service-oriented_architecture)

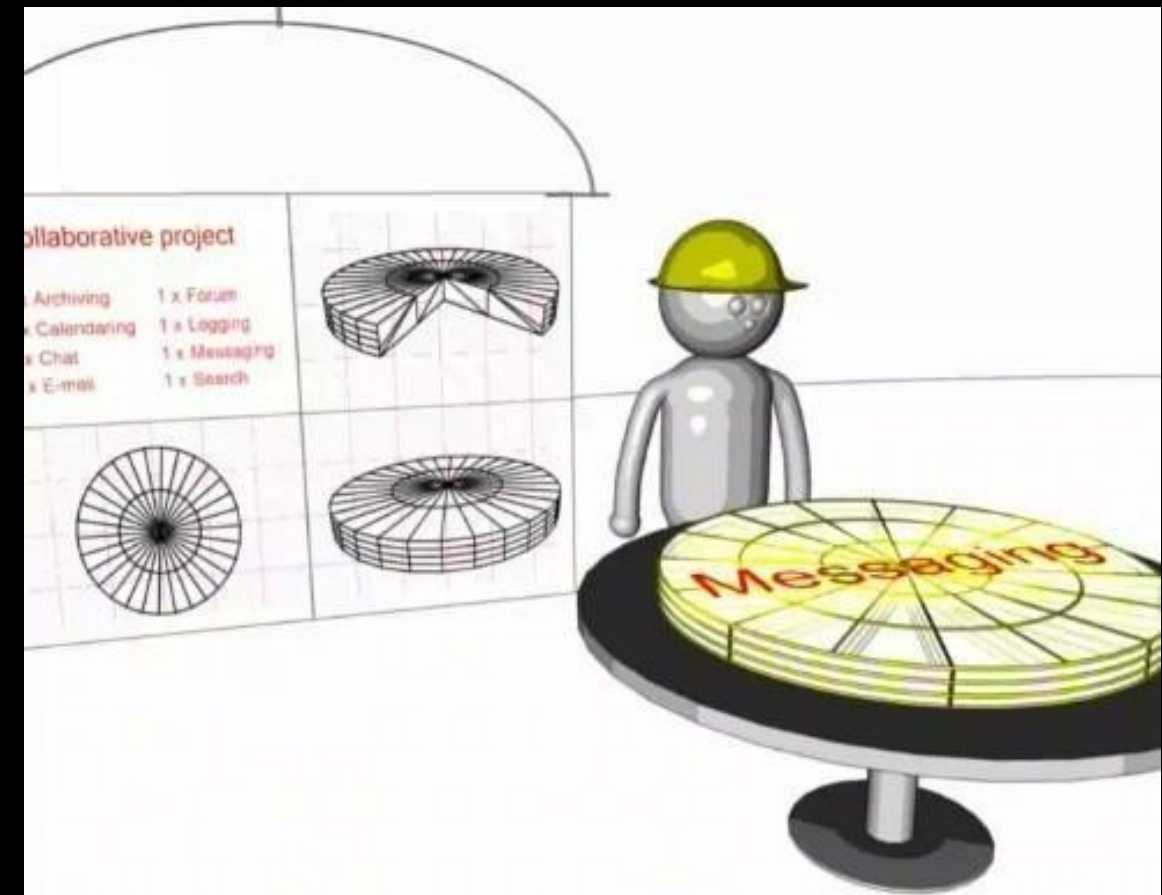
# Service Oriented Approach

- 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

# APIs


[http://en.wikipedia.org/wiki/Web\\_services](http://en.wikipedia.org/wiki/Web_services)




# There Are Many APIs

- There are organizations that put up public APIs and sell access to those APIs
- We will explore a geocoding API based on the OpenStreetMap data
- You need an account to access this API
- There is a free level of requests over time
- You pay above that rate of usage

Maps, APIs and com | X API documentation a | X Geocoding API | Dev | X Geocoding API Playg | X Jamaican Jerk Pit - | X

← → ↻ <https://www.geoapify.com> ☆

 **Geoapify** [Home](#) [Products & Services](#) [Solutions](#) [Developers](#) [Pricing](#) [Blog](#) [LOG IN](#)


# Welcome to Geoapify Location Platform

Develop location-aware apps for your business with our user friendly APIs and location services

Need to develop a location-aware app or answer location-related questions? Geoapify is a feature-rich location platform suitable for businesses of any size. We offer maps, address and location search, route optimization, reachability analysis, geodata access, and more.

Get started quickly with our code samples and tutorials. Not sure where to start? Get in touch and we'll be happy to answer all your questions.

[GET STARTED](#) [CONTACT US](#)

Have questions?  
Leave us a message 

<https://www.geoapify.com/>

API documentation and Playgro... Geocoding API | Developer Doc... +


← → ↻ https://apidocs.geoapify.com ☆

Geoapify API Docs Get API key API Documentation Code samples Go to Geoapify.com

# Geoapify Location Platform

API Documentation, Playgrounds, code samples

Glad you're here! Check out our sample code, quick start guides, and API reference documentation. You can try our APIs for free and with no registration with our Playgrounds.



## Maps

Learn all about how to add a map to your app or website

- Map tiles [Playground](#) [Docs](#)
- Static maps API [Playground](#) [Docs](#)
- Marker Icon API [Playground](#) [Docs](#)

## Addresses

Geocode addresses and locations, get address suggestions

- Geocoding API [Playground](#) [Docs](#)
- Reverse Geocoding API [Playground](#) [Docs](#)
- Address Autocomplete [Playground](#) [Docs](#)
- Batch Geocoding [Docs](#)

## Routes


Calculate and optimize routes for cars, trucks, pedestrians, and more

- Routing API [Playground](#) [Docs](#)
- Route Matrix API [Playground](#) [Docs](#)
- Map Matching API [Playground](#) [Docs](#)
- Route Planner API [Playground](#) [Docs](#)

## Reachability

Isochrones and isodistances to analyze the place's reachability

- Isolines API [Playground](#) [Docs](#)
- Geometry API [Playground](#) [Docs](#)

Have questions? Leave us a message 

<https://apidocs.geoapify.com/>

API documentation and Playgro... Geocoding API | Developer Doct...  
https://apidocs.geoapify.com/docs/geocoding/forward-geocoding/#about

Geoapify API Docs Get API key API Documentation Code samples Go to Geoapify.com

Documentation  
Addresses and location

Forward Geocoding API

- About Geocoding API
  - Getting API Key
  - API reference
  - Geocode addresses
  - Geocode cities
  - Geocode ZIP codes
  - Code samples
  - About confidence
  - Pricing
- Reverse Geocoding API
- Address autocomplete API
- Batch Geocoding API
- IP Geolocation API

## Forward Geocoding API

Geoapify provides [Geocoding REST API](#) that searches addresses worldwide. The REST API works via HTTP GET and returns JSON or XML responses. The API is cross-platform and can be used with most programming languages.

On this documentation page, you find **geocoding examples**, including calling the geocoding with **Javascript, Python, Java** languages. We show you how to geocode countries, states, cities, postcodes (or zip codes). Our code samples demonstrate how to geocode addresses with filters by geometries, bias, and different languages.

Try the API and play with Geocoding parameters in the API Playground:

[LIVE DEMO](#)

- [Authentication and Geocoding API key](#)
- [API reference](#)
- [Geocode addresses](#)
- [Geocode cities](#)
- [Geocode ZIP codes \(postcodes\)](#)
- [Code samples \(JS, NodeJS, Python, Java\)](#)
- [Pricing](#)

### Authentication and API key

To use the API, you'll need an API key. But don't worry! You can register and get a Geocod

Have questions?  
Leave us a message

<https://apidocs.geoapify.com/>

API documentation and Playc X Geocoding API | Developer D X Geocoding API Playground b X Jamaican Jerk Pit – Taste of X

← → ↻ https://apidocs.geoapify.com/playground/geocoding/#reverse ☆

**Geoapify API Docs** [Get API key](#) API Documentation Code samples [Go to Geoapify.com](#)

## Geocoding API Playground

[Documentation](#)

Enter an address to search, click on the map to set filter and bias. Learn more about [Geocoding API](#), [Reverse Geocoding API](#), or [Address Autocomplete API](#).

[Geocoding](#) Reverse geocoding Autocomplete

Text  Structured

314 South Thayer Ann Arbor, MI 48104

Options	Address level: not specified, language: not specified, number of results: not specified	▼
Filter	Not specified	▼
Bias	Not specified	▼

[Search](#)

Request [URL](#) [CURL](#) [JS](#) [Node.js](#) [Python](#) [PHP](#) [Java](#)

https://api.geoapify.com/v1/geocoding/search?text=314+South+Thayer+Ann+Arbor,+MI+48104

Have questions? Leave us a message

<https://apidocs.geoapify.com/>

Browser tabs: Maps, APIs and... API document: Geocoding API Geocoding X Geocoding API Jamaican Jerk py4e-data.dr-chuc

Address bar: <https://apidocs.geoapify.com/docs/geocoding/forward-geocoding/#pricing>

Geoapify API Docs [Get API key](#) API Documentation Code samples [Go to Geoapify.com](#)

Documentation  
Addresses and location

Forward Geocoding API

- About Geocoding API
- Getting API Key
- API reference
- Geocode addresses
- Geocode cities
- Geocode ZIP codes
- Code samples
- About confidence
- Pricing
- Reverse Geocoding API
- Address autocomplete API
- Batch Geocoding API
- IP Geolocation API

```
32 | .catch(error => console.log('error', error));
```

## Pricing

Geoapify Location Platform provides APIs which have different difficulty, execution times and require different resource capacities on our servers.

To make our pricing plans easy-to-understand and unify them we introduced "credits" currency that is used to describe conditions and options of each pricing plan. All the credits used for Geoapify API calls per 24 hours accumulated to Daily API usage.

Check [Geoapify Pricing Plans](#) and choose the one that fits your needs the best.


One Geocoding API / Reverse Geocoding API / Autocomplete API request is equal to one credit:

API name	Cost in credits	Example
Geocoding API	1 request = 1 credit	100 requests costs 100 credits
Reverse Geocoding API	1 request = 1 credit	100 requests costs 100 credits
Address Autocomplete	1 request = 1 credit	100 requests costs 100 credits

You can save up to 50% of API call costs when you send [Batch Geocoding requests](#).

Learn more

- [Geoapify Pricing Plans](#)
- [Pricing Details](#)

Have questions?  
Leave us a message 

<https://apidocs.geoapify.com/>

Browser tabs: Maps, APIs and... API document: Geocoding AP... Geocoding X Geocoding AP Jamaican Jerk py4e-data.dr-chuc

Address bar: <https://apidocs.geoapify.com/docs/geocoding/forward-geocoding/#quick-start>

Geoapify API Docs [Get API key](#) API Documentation Code samples [Go to Geoapify.com](#)

Documentation  
Addresses and location

Forward Geocoding API

- About Geocoding API
- Getting API Key**
- API reference
- Geocode addresses
- Geocode cities
- Geocode ZIP codes
- Code samples
- About confidence
- Pricing

Reverse Geocoding API

Address autocomplete API

Batch Geocoding API

IP Geolocation API

## Authentication and API key

To use the API, you'll need an API key. But don't worry! You can register and **get a Geocoding API key for free** without a credit card. Our free plan includes up to 3000 geocoding requests/day. For more information on our plans, visit the [Pricing page](#).

### How to get Geocoding API key

1. Register on [Geoapify MyProjects](#) page
2. Create a new project.
3. Go to the API Keys section. One API key is generated automatically. You can generate multiple API keys per project if required.
4. Optionally, you can protect the API key by listing allowed IP addresses, HTTP referrers, origins, and CORS.
5. Choose "Geocoding API" and an API key to get an URL and programming code.
6. Press the "Try" button to execute the API call and get the result object.

### API reference

**The Geocoder API accepts both structured and free-form addresses as an input and returns JSON, GeoJSON, and XML objects as a response. In addition, you can specify location filters and preferred geographical areas to make the address search more accurate and focused. #### Request URL**

```
https://api.geoapify.com/v1/geocode/search?REQUEST_PARAMS
```

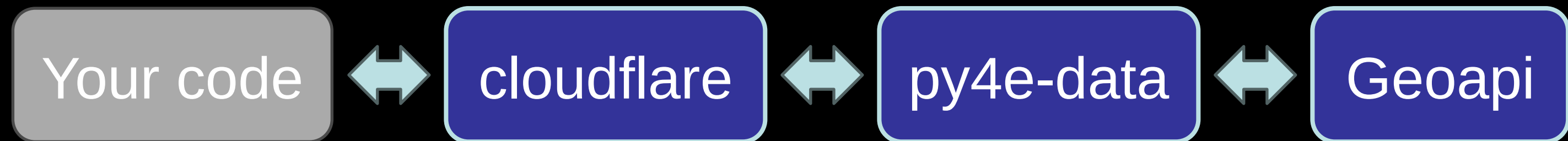
Here are Geocoding URL examples (click on a URL to test the API):

Have questions? Leave us a message

<https://apidocs.geoapify.com/>

# An API Proxy

- To avoid making you get an account, I have a well-hidden web server that acts as a proxy for the Geoapify data
- This proxy does not require a password – but it does have rate limits and is heavily cached using an edge-caching service for performance



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "datasource": {
          "sourcename": "openstreetmap",
          "attribution": "© OpenStreetMap contributors",
          "license": "Open Database License",
          "url": "https://www.openstreetmap.org/copyright"
        },
        "country": "United States",
        "country_code": "us",
        "state": "Michigan",
        "county": "Washtenaw County",
        "city": "Ann Arbor",
        "lon": -83.7312291,
        "lat": 42.2681569,
        "state_code": "MI",
        "result_type": "city",
        "formatted": "Ann Arbor, MI, United States of America",
```

<http://py4e-data.dr-chuck.net/opeengeo?q=Ann+Arbor%2C+MI>

Note, for this course, we operate through a proxy of the geoapi data to avoid rate limitation and authentication.

**opeengeo.py**

```
import urllib.request, urllib.parse
import http, json, ssl

serviceurl = 'https://py4e-data.dr-chuck.net/opengeo?'

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

    address = address.strip()
    parms = dict()
    parms['q'] = address

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

    print('Retrieving', url)
    uh = urllib.request.urlopen(url, context=ctx)
    data = uh.read().decode()
    print('Retrieved', len(data), 'characters', data[:20].replace('\n', ' '))

    js = json.loads(data)

    lat = js['features'][0]['properties']['lat']
    lon = js['features'][0]['properties']['lon']
    print('lat', lat, 'lon', lon)
    location = js['features'][0]['properties']['formatted']
```

Enter location: Ann Arbor, MI  
Retrieving https://py4e-data.  
dr-chuck.net/opengeo?q=Ann+Arbor%2C+MI  
Retrieved 1319 characters {"type":"FeatureColl  
lat 42.2681569 lon -83.7312291  
Ann Arbor, MI, United States of America

**opengeo.py**

# Summary

- Service Oriented Architecture - allows an application to be broken into parts and distributed across a network
- An Application Program Interface (API) is a contract for interaction
- Web Services provide infrastructure for applications cooperating (an API) over a network - SOAP and REST are two styles of web services
- XML and JSON are serialization formats



# Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and [open.umich.edu](http://open.umich.edu) and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors here