Python Object-Oriented Program with Libraries

Unit 3: Web Programming

CHAPTER 4: WEB APIS

DR. ERIC CHOU

IEEE SENIOR MEMBER

The three basic types of APIs

APIs take three basic forms: local, web-like and program-like. Here's a look at each type.

Local APIs

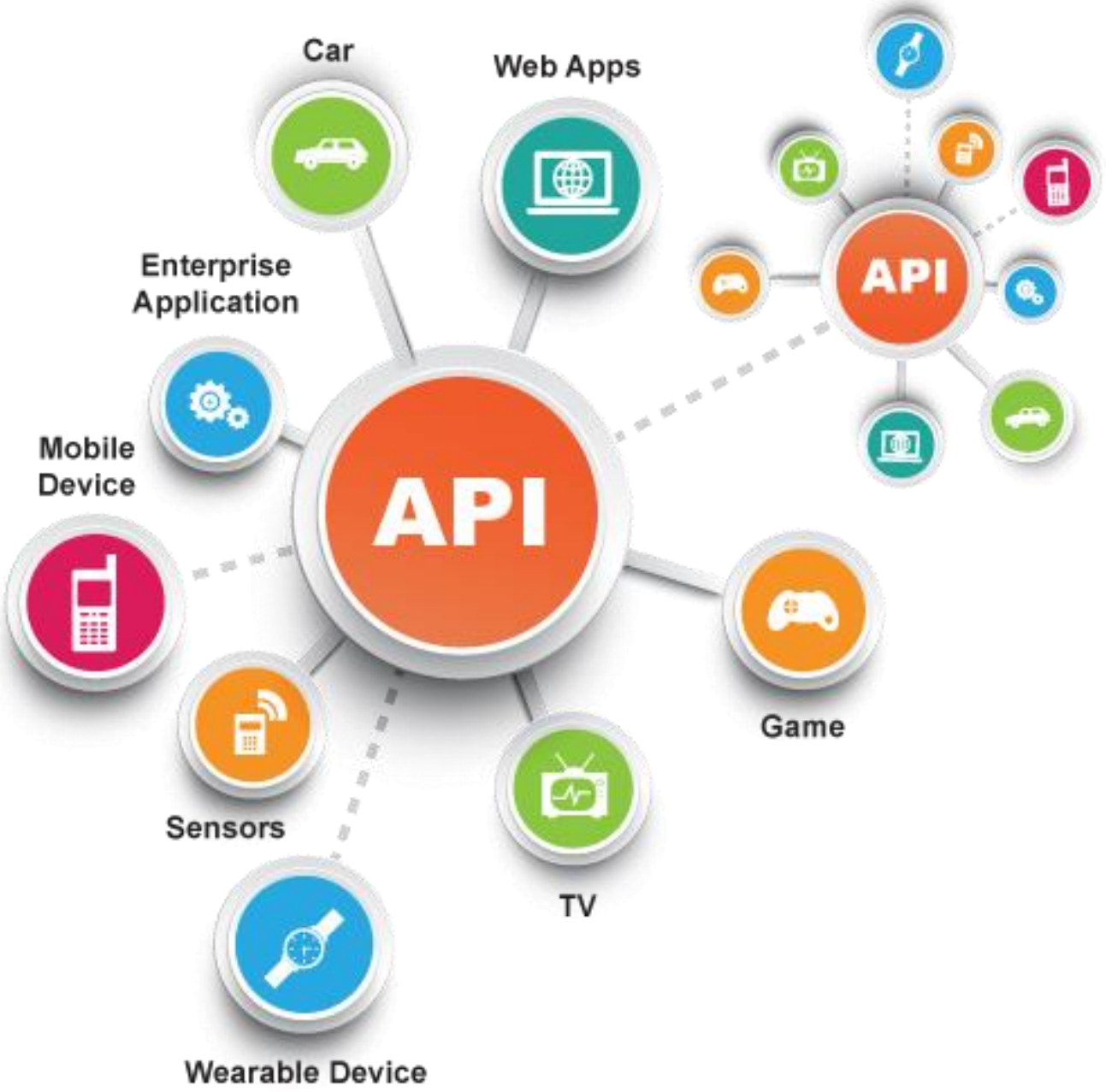
The original API, created to provide operating system or middleware services to application programs.

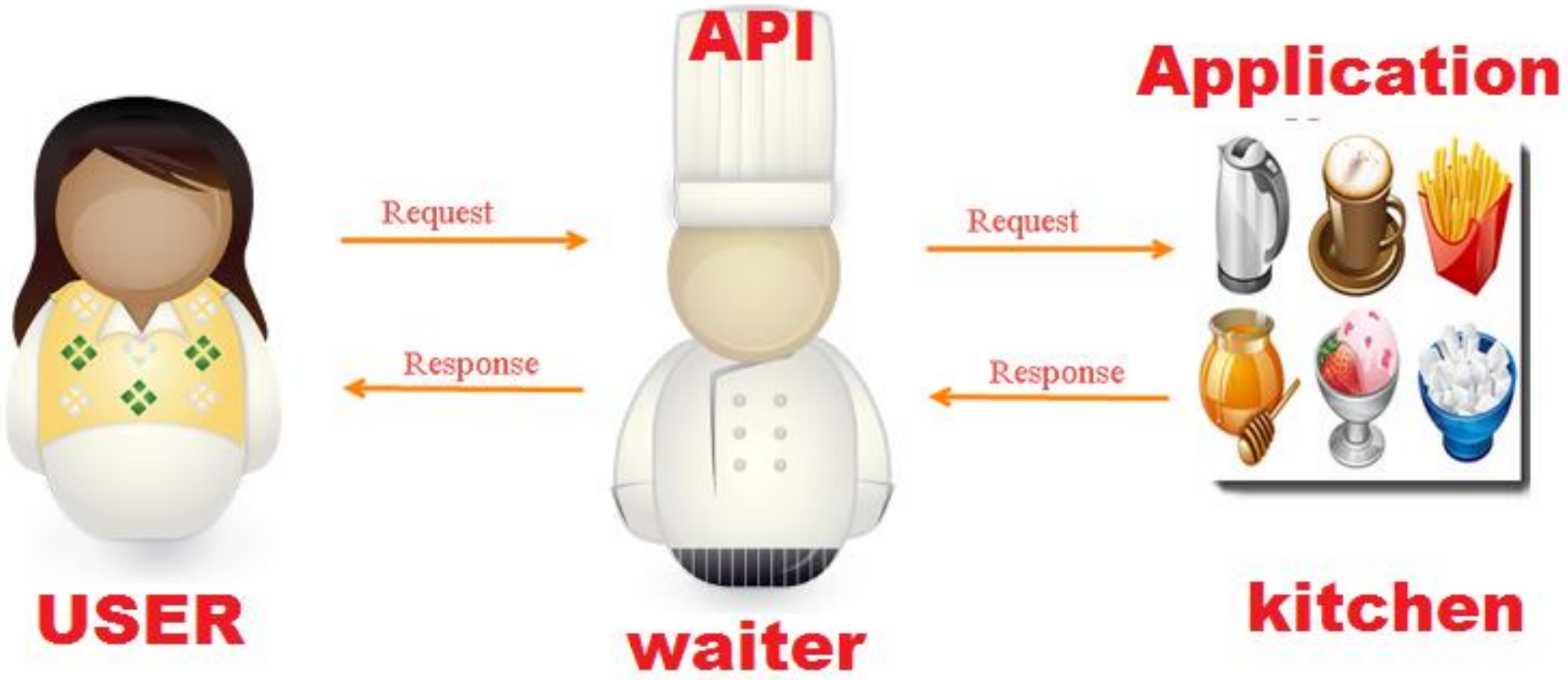
Web APIs

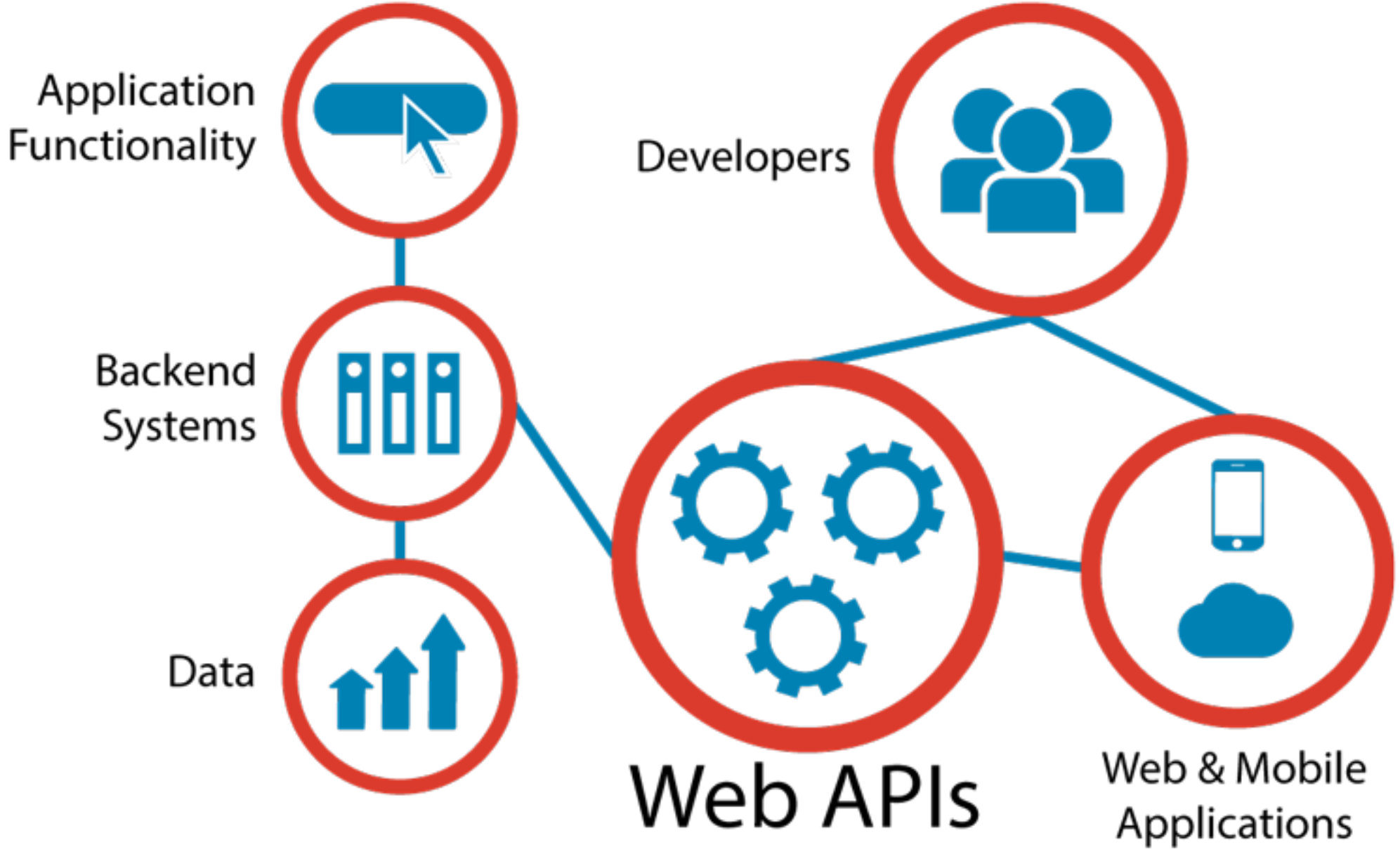
Designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Often called REST APIs or RESTful APIs.

Program APIs

Based on RPC technology that makes a remote program component appear to be local to the rest of the software.


Car
Web Apps
Enterprise Application
Mobile Device
API
API
Game
Sensors
TV
Wearable Device


API
Application
Request
Request
Response
Response
USER
waiter
kitchen


Application Functionality
Developers
Backend Systems
Data
Web APIs
Web & Mobile Applications

Overview

LECTURE 1

(81) lakers clippers - You

Secure | https://www.youtube.com/results?search_query=lakers+clippers

Apps | S | EC Live | De Anza | BB | UCI | Wyz | Ant | T | Other bookmarks

YouTube

lakers clippers

About 493,000 results

FILTER

Los Angeles Lakers vs LA Clippers Full Game Highlights / Week 1 / 2017 NBA Season

MLG Highlights 701K views • 3 weeks ago

Los Angeles **Lakers** vs LA **Clippers** Full Game Highlights / Week 1 / 2017 NBA Season For more information, as well as all the ...

Los Angeles Lakers vs LA Clippers 1st Half Highlights / Week 1 / 2017 NBA Season

MLG Highlights 146K views • 3 weeks ago

Los Angeles **Lakers** vs LA **Clippers** 1st Half Highlights / Week 1 / 2017 NBA Season For more information, as well as all the latest ...

Protocol

Resource

http://www.youtube.com/results?search_query=lakers+clippers

Host Name

Query

Client Side

web pages (Browser Engine)

web APIs

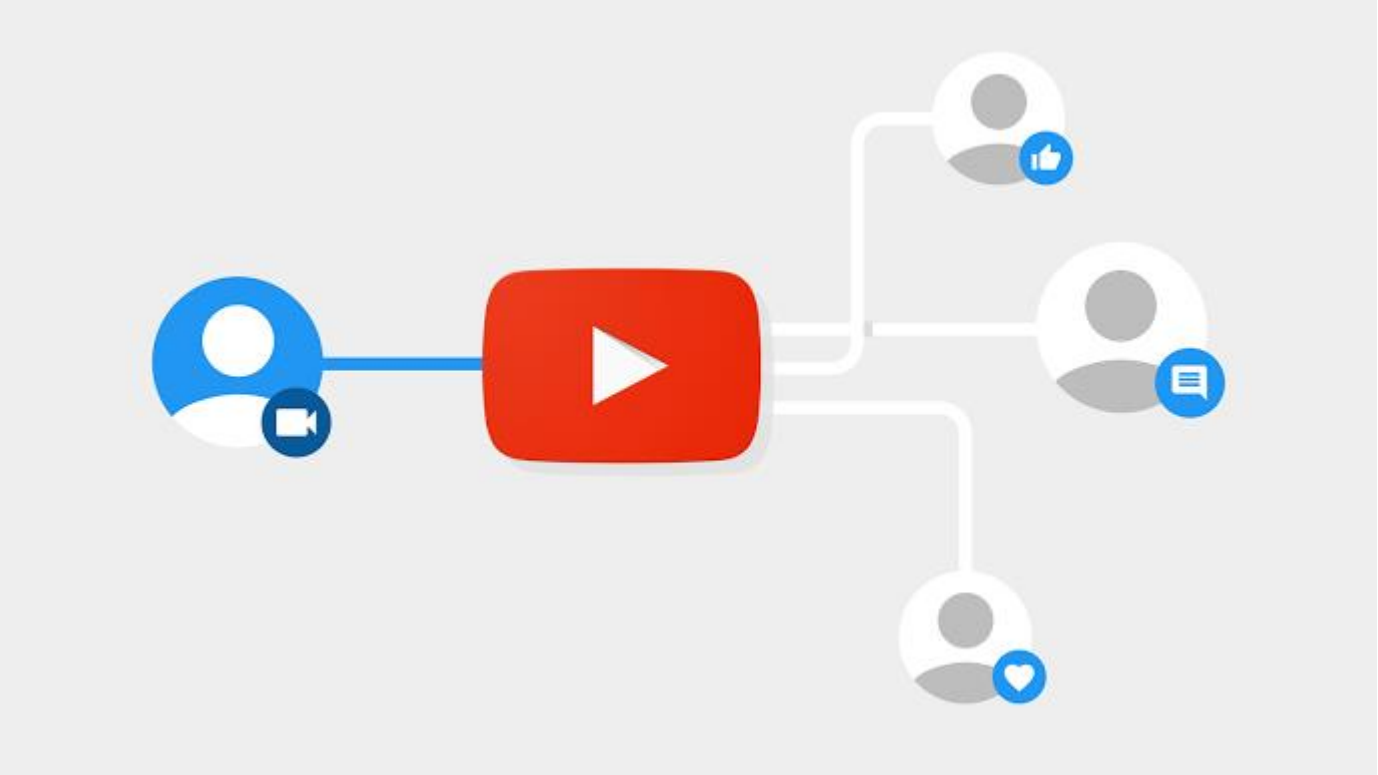
Server Side: Servlet, Server Program
Node.js, Java EE, Python, and …

VS

Application Programming Interfaces (API)

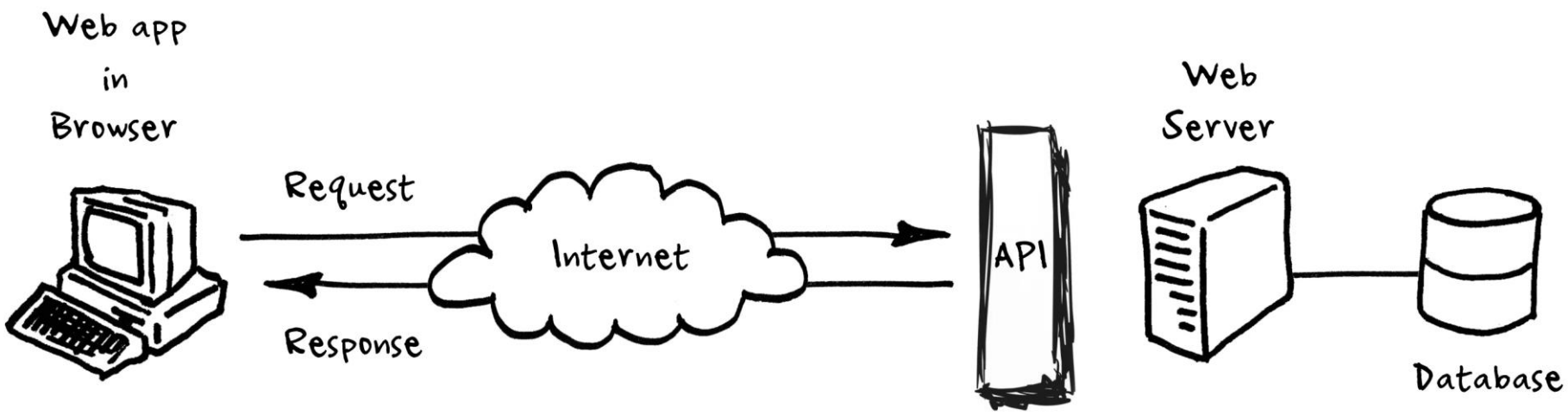
- Web-site Server’s Programming Interfaces to an Application.
- The **YouTube Data API** is a **web API**, meaning that a program interacts with it by sending an HTTP request — just like downloading a web page — and gets its answer back as an HTTP response.
- The **URL** specifies not only the operation we want to perform, but also the parameters for that operation (e.g., the search query).
- The response is formatted in a way that's structured so that it will be easy for a program to parse and understand, in a format that's published, so you can rest assured that it won't change when YouTube periodically changes the look of their web pages for human users.

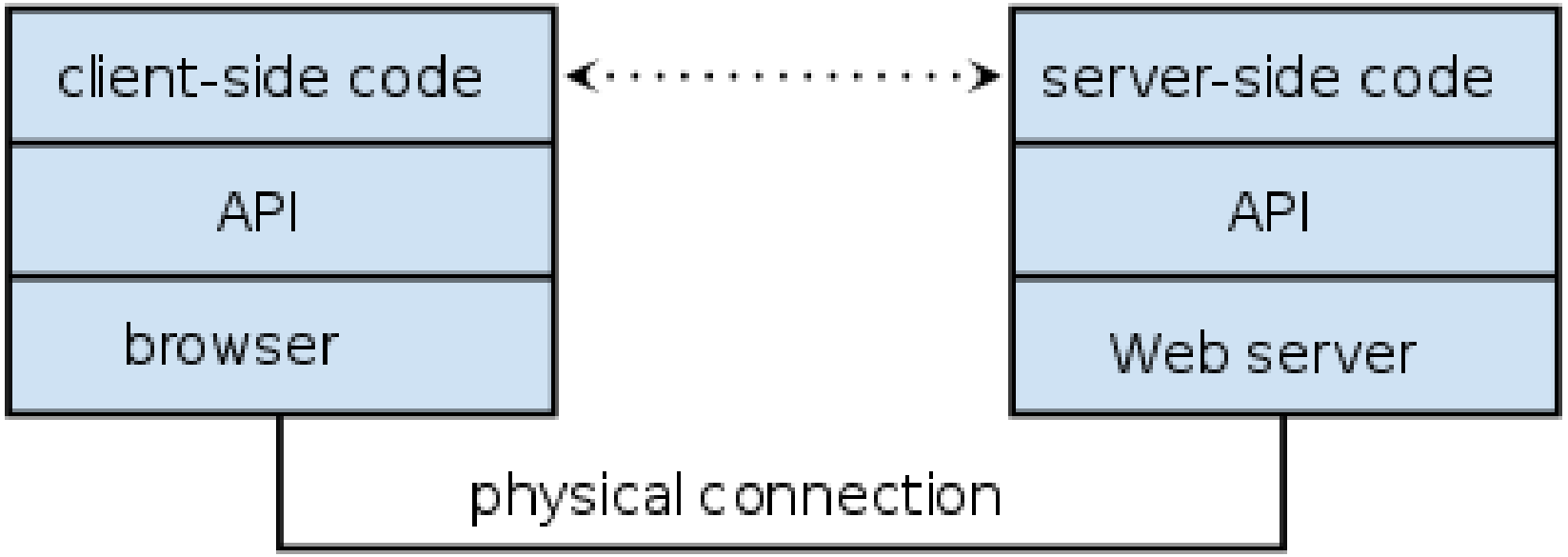
YouTube > Data API



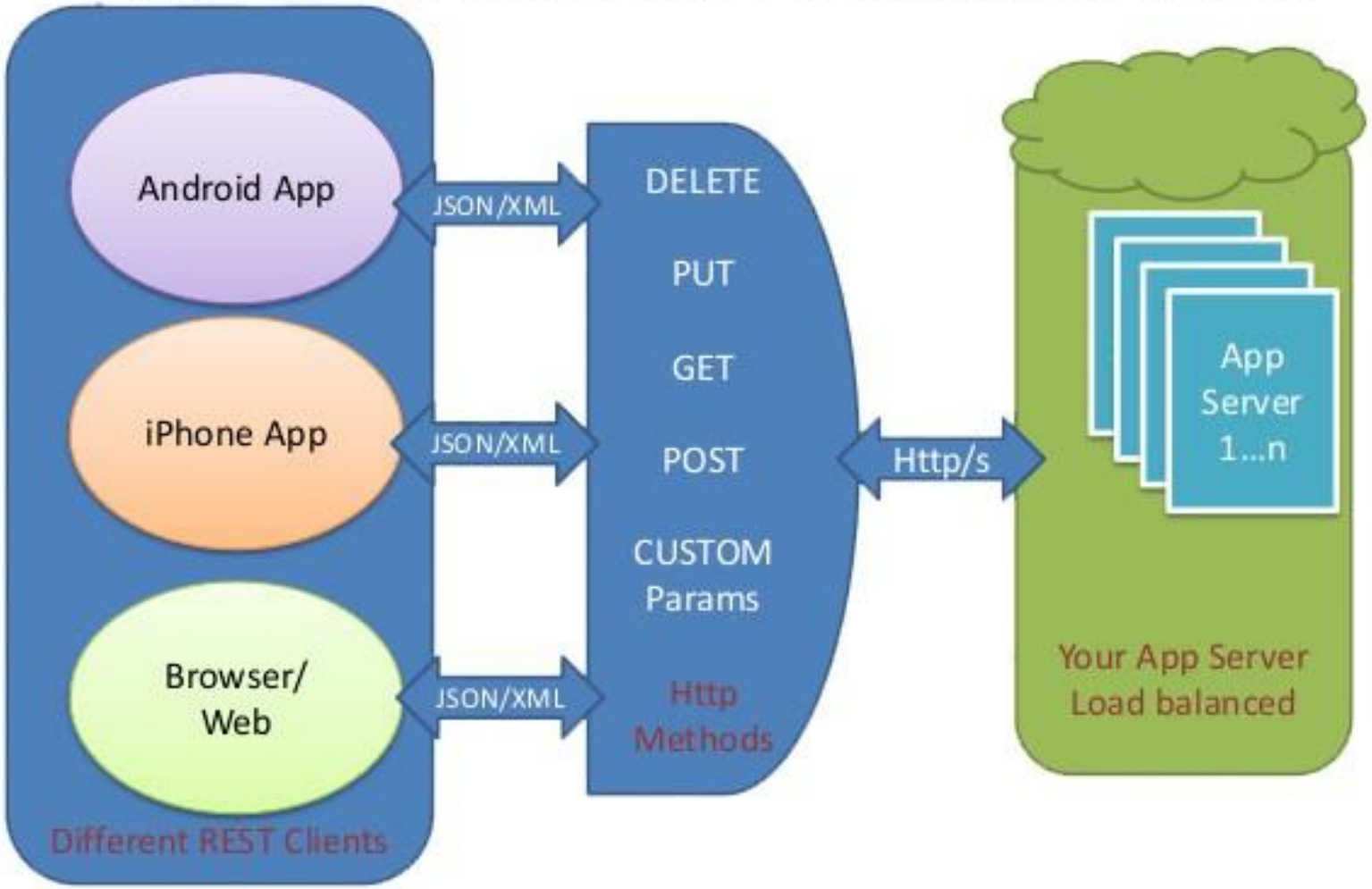
Python Sample Code uses API

```python
#!/usr/bin/python

import httplib2
import os
import re
import sys

from apiclient.discovery import build
from apiclient.errors import HttpError
from oauth2client.client import flow_from_clientsecrets
from oauth2client.file import Storage
from oauth2client.tools import argparser, run_flow


# The CLIENT_SECRETS_FILE variable specifies the name of a file that contains
# the OAuth 2.0 information for this application, including its client_id and
# client_secret. You can acquire an OAuth 2.0 client ID and client secret from
# the {{ Google Cloud Console }} at
# {{ https://cloud.google.com/console }}.
# Please ensure that you have enabled the YouTube Data API for your project.
# For more information about using OAuth2 to access the YouTube Data API, see:
#   https://developers.google.com/youtube/v3/guides/authentication
# For more information about the client_secrets.json file format, see:
#   https://developers.google.com/api-client-library/python/guide/aaa_client_secrets

CLIENT_SECRETS_FILE = "client_secrets.json"

# This variable defines a message to display if the CLIENT_SECRETS_FILE is
# missing.
MISSING_CLIENT_SECRETS_MESSAGE = """
WARNING: Please configure OAuth 2.0
```


Web app
in
Browser
Request
Response
Internet
API
Web
Server
Database


client-side code
API
browser
server-side code
API
Web server
physical connection

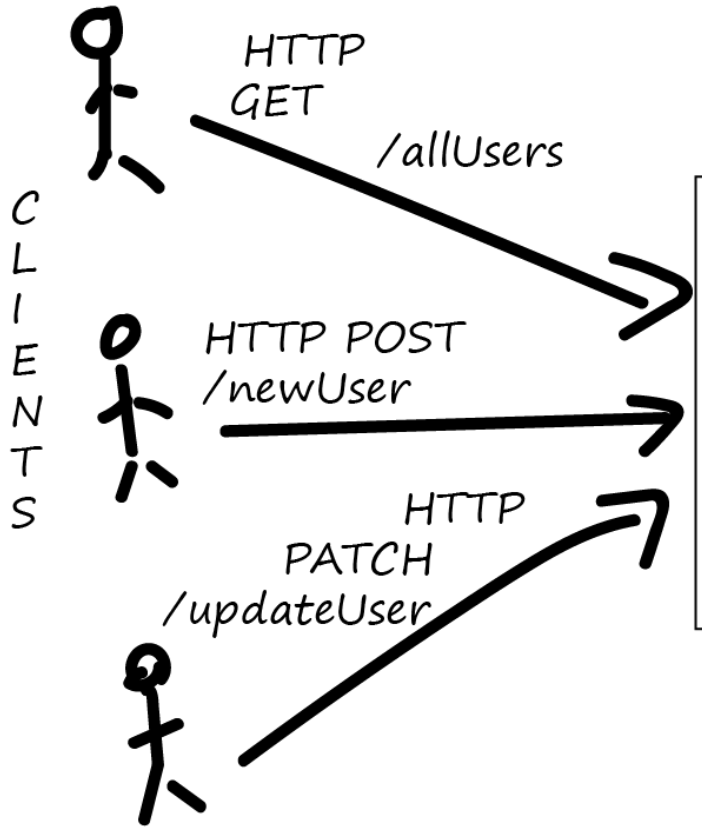
REST API Architecture



Note: REST is an Open API development framework

Rest API Basics

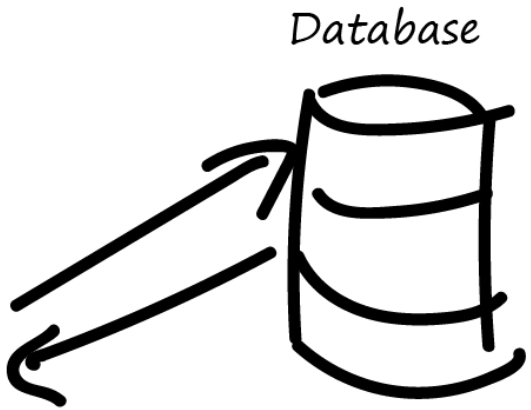
Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database



Our Clients, send HTTP Requests and wait for responses

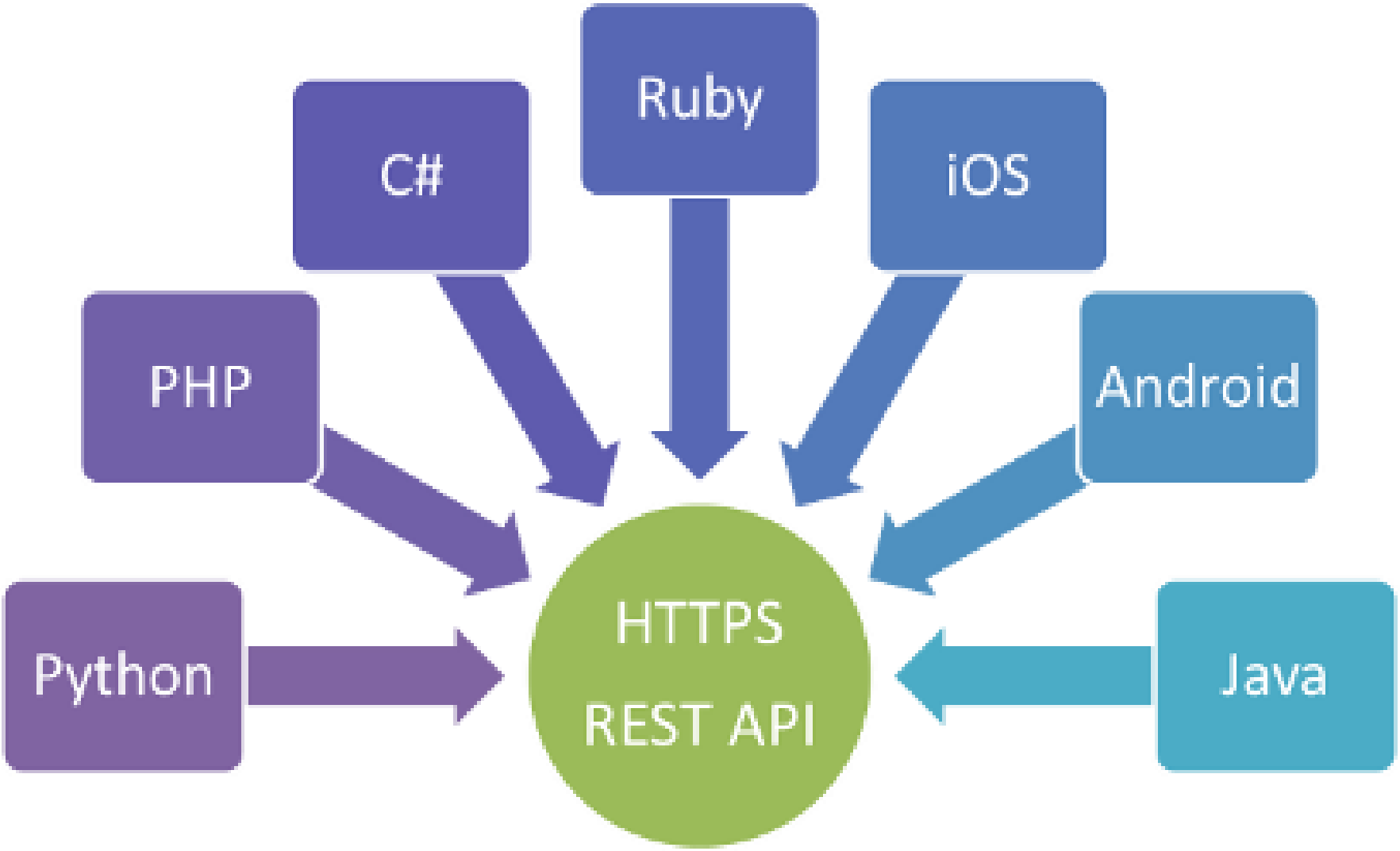
Rest API

Recieves HTTP requests from Clients and does whatever request needs. i.e create users



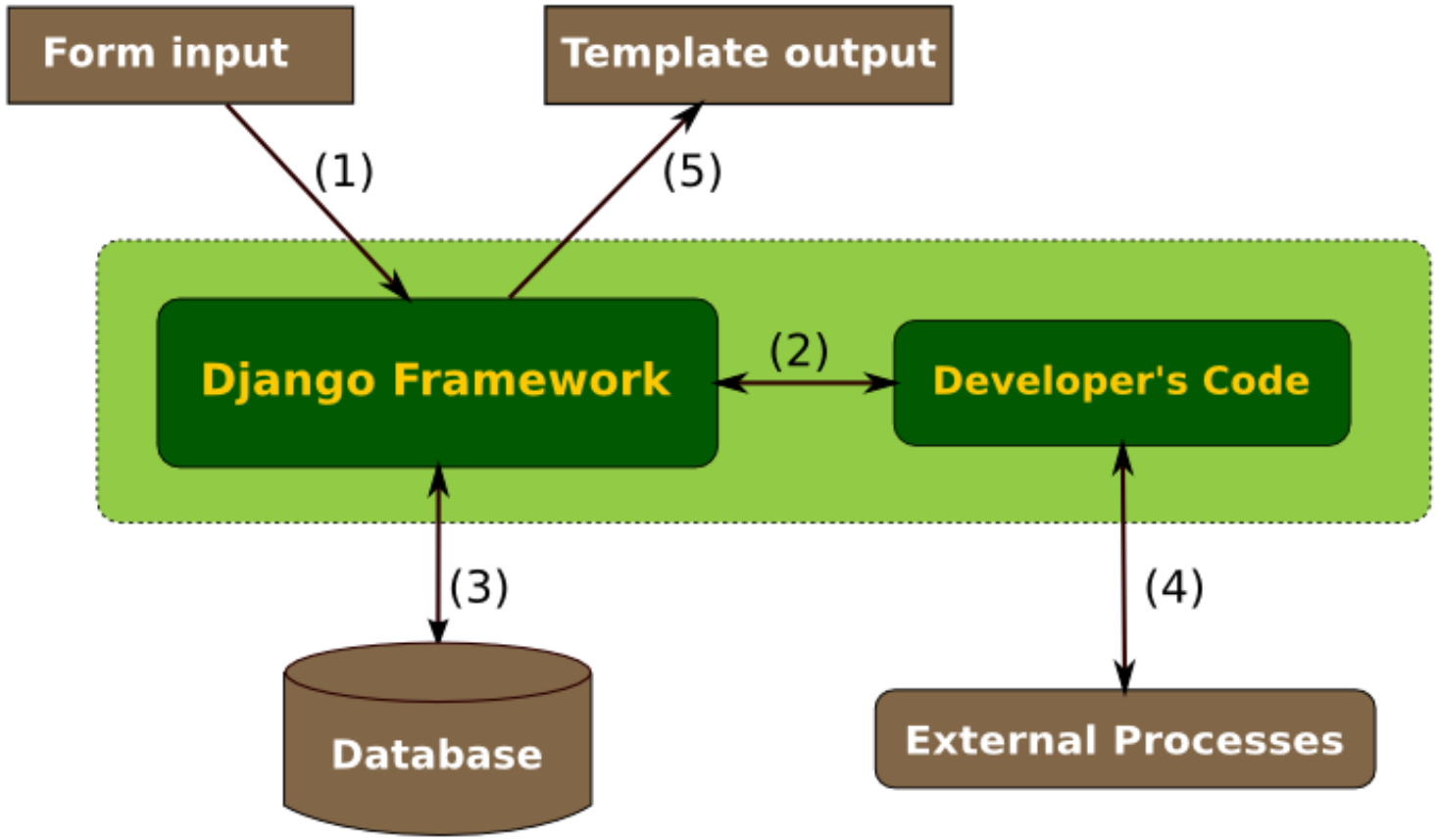
Our Rest API queries the database for what it needs

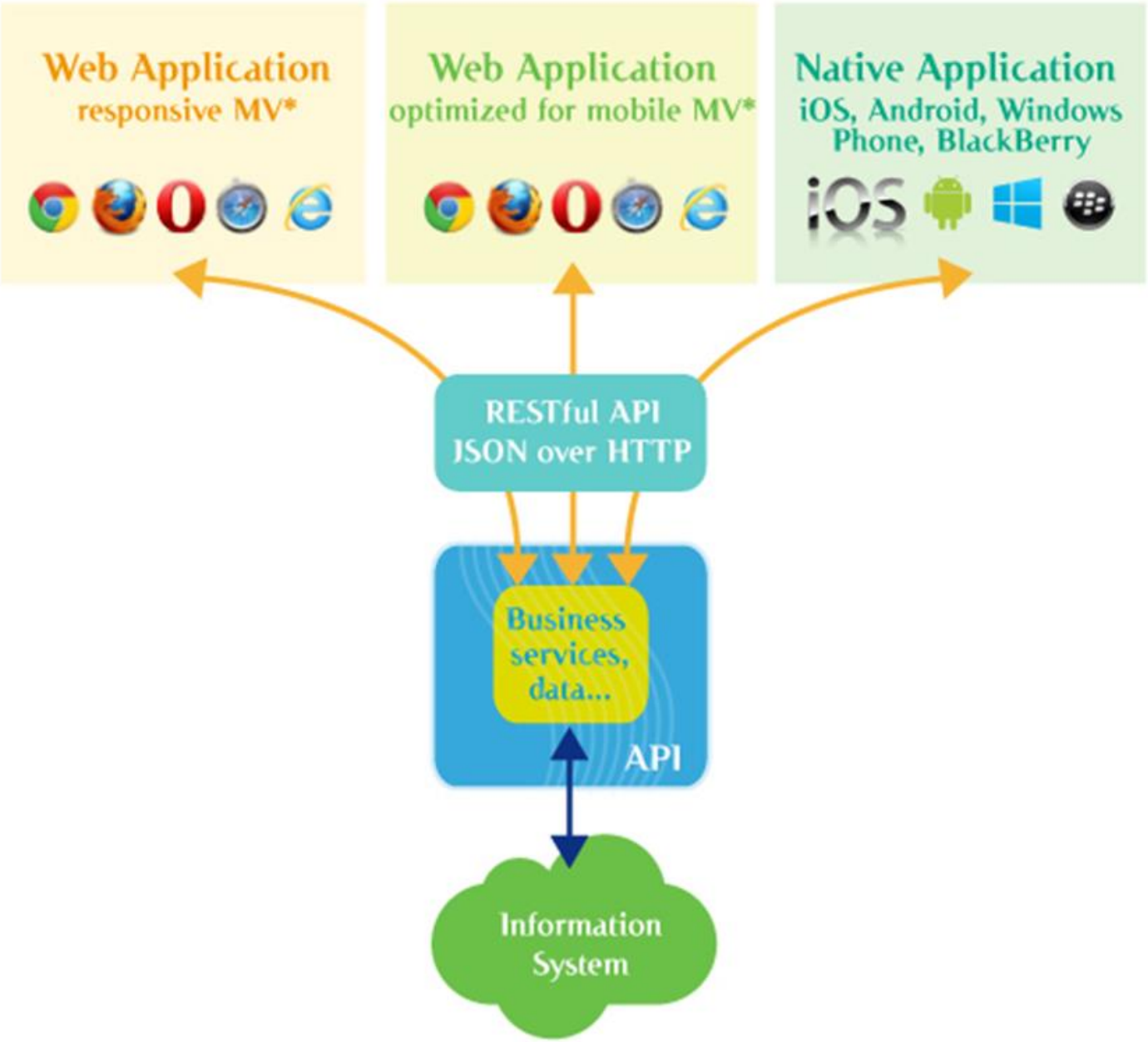
Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.


Ruby
C#
iOS
PHP
Android
Python
HTTPS
REST API
Java

Tools to build API for Python:

- Python Language
- HTTP
- Django
- JSON
- REST
- SQL (MySQL and others)
- Client-Side App Design
- Flask
- Panda




Web Application
responsive MV*
Web Application
optimized for mobile MV*
Native Application
iOS, Android, Windows Phone, BlackBerry
iOS
RESTful API
JSON over HTTP
Business services, data...
API
Information System

The API Landscape

Last Update: March, 2017

Business Processes as an API/API-as a Product/Transactional APIs (151)

- SMS / Voice APIs (22)
- Fax / Mail (9)
- Email APIs (20)
- Search Engine APIs (5)
- Payments APIs (25)
- Accounting API (10)
- eCommerce APIs (4)
- AI APIs (4)
- Data-as-a-service/Messaging (50)
- Online Forms(2)
- Humans as a Service APIs (2)
- HR APIs (2)

API Lifecycle platform (78)

- API Documentation Platform (6)
- API Testing (6)
- API Management (16)
- API Analytics / Monitoring (5)
- Streaming APIs (9)
- Developer portal (5)
- IOT Platforms/Networks/Devices API (7)
- Access level and Identity management (20)
- API Oriented IT Consulting Firms (8)

Backend Building Tools/MBaaS (14)

- MBaaS (7)
- GraphQL BaaS (3)
- DBaaS (4)

API Abstraction / Integration Platform (32)

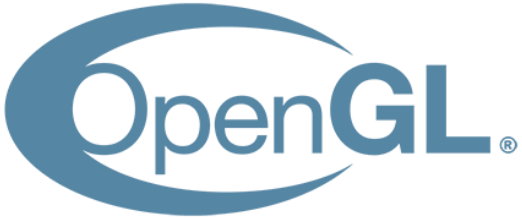
- Horizontal API abstractions (5)
- Automation/ Orchestration (8)
- Vertical API Abstractions (13)
- Domotics (6)

Library (1970-1990)

Foundation Class (1990-2000)

Design Framework (2000-Present)
RTE (Run-time Environment)

Web-APIs
(2010-Future)



Advancement of Re-useable/Service Code

Standards for web APIs: URL Query

LECTURE 1

URLs with Query Parameters

For many years, there has been a standard for URLs that include these kinds of parameters, which are called query parameters. A hypothetical example of a URL with query parameters follows:

http://www.blah.com/some/page?param1=value1&param2=value2&param3=value3

- Before the **?** character, this looks just like any other URL.
- The **?** is special; it indicates that what follows it will be a sequence of **query parameters**.
- Each parameter is specified as a **name** and a **value**, with an **=** separating them; the parameters themselves are separated by **&** characters. ({name:value} is dictionary or JSON format)

Demo Program: amazon.py

Query Parameters:

- **url** : whose value is some kind of URL, though it's not clear exactly what it's being used for

and

- **field-keywords** : which appears to be my original query, with the spaces mysteriously replaced with + characters.

We would have to know more about how Amazon's web site is implemented to know for sure what the query parameters mean, but we can sometimes suss out their meaning just by looking at them.

Fetching URLs

The simplest way to use **urllib.request** is as follows:

```
import urllib.request
response = urllib.request.urlopen('http://python.org/')
html = response.read()
```

- Many uses of urllib will be that simple (note that instead of an ‘http:’ URL we could have used an URL starting with ‘ftp:’, ‘file:’, etc.).
- However, it’s the purpose of this tutorial to explain the more complicated cases, concentrating on HTTP.

Fetching URLs

- HTTP is based on requests and responses - the client makes requests and servers send responses.
- urllib.request mirrors this with a **Request** object which represents the HTTP request you are making.
- In its simplest form you create a Request object that specifies the **URL** you want to fetch. Calling **urlopen** with this **Request** object returns a response object for the URL requested.
- This response is a file-like object, which means you can for example call **.read()** on the response:

```
import urllib.request
req = urllib.request.Request('http://www.voidspace.org.uk’)
response = urllib.request.urlopen(req)
the_page = response.read()
```

module handling utf8 and other file formats

header setting to pass the auto-web-bot check

Get the qualified URL request

utf8 string in response message

Write the response to a local html file

```
import urllib.request
import codecs

# prepare API request
query_parameters = "url=search-alias%3Daps&field-keywords=u2+the+joshua+tree"
url = "https://www.amazon.com/s/ref=nb_sb_noss_2/183-7159112-3775704?"
url_query = url+query_parameters

# headers needed to pass the web-site auto-bot check.
headers = {}
headers['User-Agent'] = "Mozilla/5.0 (X11; Linux i686)"  # server type at google

# make API request
request = urllib.request.Request(url_query, headers=headers) # make request to google.com
response = urllib.request.urlopen(request)   # just like open a file
response_data = response.read()
response_text = response_data.decode("utf8")  # convert bytes to utf8

# convert text to a html file

fp = codecs.open("joshua_tree.html", "w", "utf8")   # open a file for utf-8 text format
fp.write(response_text)
fp.close()

response.close()
```

C:/Eric_Chou/Python Course/Python Object-Oriented Programming with Libraries/PyDev/U3 Network/HttpAndURL/WebAPI/joshua_tree.html (Bison3) - Brackets

File Edit Find View Navigate Debug Help

Working Files

buildBH01.bat
Exercise75.java
Lesson7_1.java
java-generic-programming-pa
Exercise86.java
fileIO.java — AP2017-18
CircleTester.java
Name.h
joshua_tree.html

Bison3

build1.bat
calc3.h

```
1
2 <!doctype html><html class="a-no-js" data-19ax5a9jf="dingo"><head><script>var
  aPageStart = (new Date()).getTime();</script><meta charset="utf-8">
3 <script type='text/javascript'>var ue_t0=ue_t0||+new Date();</script>
4 <script type='text/javascript'>
5 var ue_csm = window,
6     ue_hob = +new Date();
7 (function(d){var e=d.ue=d.ue||{},f=Date.now||function(){return+new
  Date};e.d=function(b){return f()-(b?0:d.ue_t0)};e.stub=function(b,a){if(!b[a]){var c=
  [];b[a]=function()
  {c.push([c.slice.call(arguments),e.d(),d.ue_id])};b[a].replay=function(b){for(var
  a;a=c.shift();)b(a[0],a[1],a[2])};b[a].isStub=1}};e.exec=function(b,a){return
  function(){if(1==window.ueinit)try{return b.apply(this,arguments)}catch(c)
  {ueLogError(c,{attribution:a||"undefined",logLevel:"WARN"})}}}})(ue_csm);
```

Line 1, Column 1 — 3985 Lines | INS | UTF-8 | HTML | Spaces: 4

Saved html file in bracket editor

Amazon.com: u2 the josh

file:///C:/Eric_Chou/Python%20Course/Python%20Object-Oriented%20Programming%20with%20Libraries/PyDev/U3%20Network/HttpAndURL/WebAPI/joshua_tree.h...

NEW & INTERESTING FINDS ON AMAZON EXPLORE

amazon Try Prime | All | u2 the joshua tree | Countdown to black friday FREE Shipping

Departments | Browsing History | Your Amazon.com | Today's Deals | Gift Cards & Registry | EN | Hello. Sign in Account & Lists | Orders | Try Prime | 0 Cart

1-16 of 488 results for "u2 the joshua tree"

Sort by Relevance

Show results for

CDs & Vinyl
Rock
Pop
Alternative Rock
Post-Punk
Adult Alternative
See more

Digital Music
Rock
See All 21 Departments

SPONSORED BY UNIVERSAL MUSIC ENTERPRISES

U2 – The Joshua Tree 30 Years

›Shop now

The Joshua Tree Super Deluxe (4 C...

The Joshua Tree Super Deluxe (7 L...

Ad feedback

Sponsored

U2 Joshua Tree Tour 1987 Three Rivers Stadium Live Concert Retro Art Print — Poster Size — Print of Retro Concert Poster — Features Bono, The Edge, Adam Clayton and Larry Mullen, Jr..

File open in Chrome from bracket

Demo Program: amazon2.py

Go PyCharm!!!

amazon2.py

```python
import urllib.request
import urllib.parse
import codecs

# prepare API request
query_parameters_strings = {'url':'search-alias%3Daps', 'field-keywords':'u2 the joshua tree'}
query_parameters = urllib.parse.urlencode(query_parameters_strings)
url = "https://www.amazon.com/s/ref=nb_sb_noss_2/183-7159112-3775704?"
url_query = url+query_parameters

# headers needed to pass the web-site auto-bot check.
headers = {}
headers['User-Agent'] = "Mozilla/5.0 (X11; Linux i686)"  # server type at google

# make API request
request = urllib.request.Request(url_query, headers=headers) # make request to google.com
response = urllib.request.urlopen(request)   # just like open a file
response_data = response.read()
response_text = response_data.decode("utf8")  # convert bytes to utf8

# convert text to a html file
lines = response_text.splitlines()
fp = codecs.open("joshua_tree2.html", "w", "utf8")   # open a file for utf-8 text format
for line in lines:
    fp.write(line+"\n")
fp.close()
response.close()
```

Dictionary for parameter list

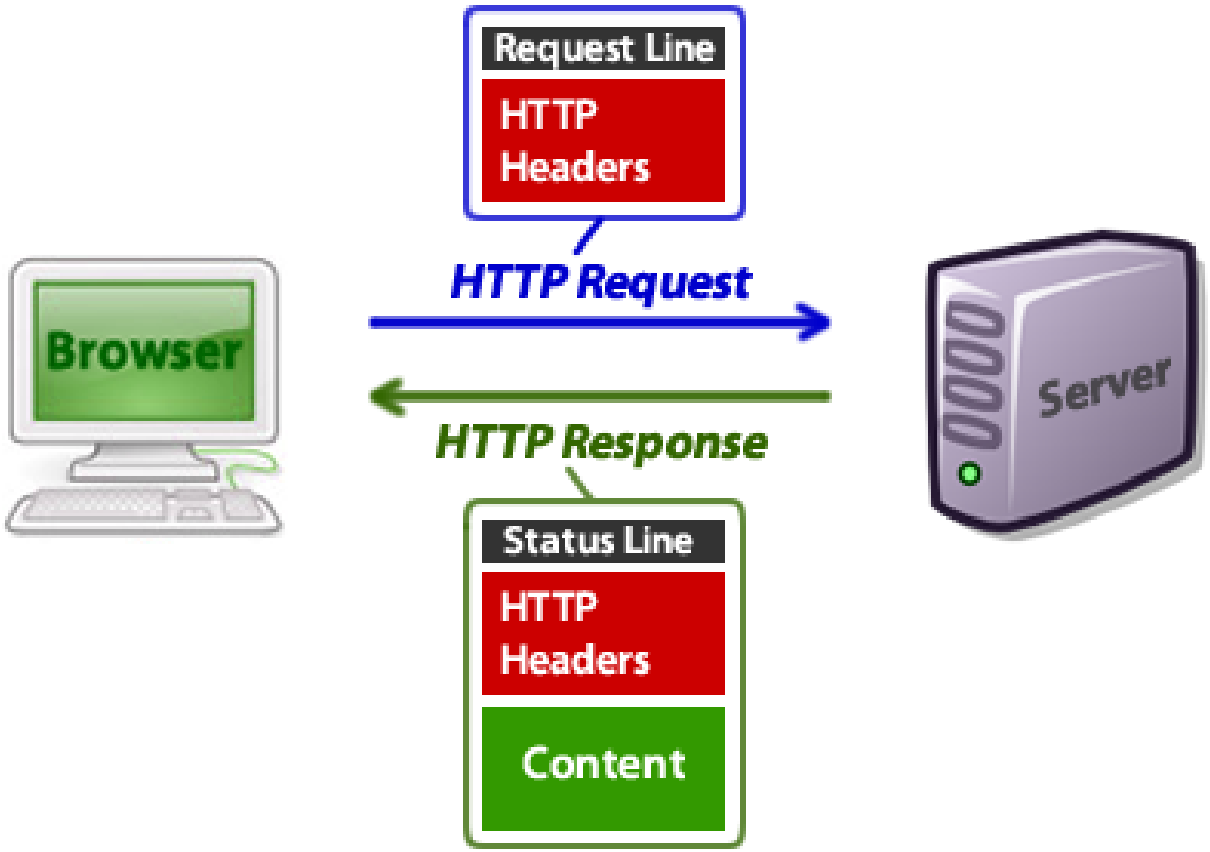
Separated by space not + sign

Convert from dictionary to query parameter string

Convert line by line with “\n” to make the html more readable

Standards for web APIs: header

LECTURE 1


Request Line
HTTP Headers
HTTP Request
Browser
Server
HTTP Response
Status Line
HTTP Headers
Content

Post Method: Request Object (no Data request for GET, Data request for POST)

CGI Program: .php, index.html, …

Protocol

GET /tutorials/other/top-20-mysql-best-practices/ **HTTP/1.1**

Host: net.tutsplus.com

header fields

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120

Pragma: no-cache

Cache-Control: no-cache

useragentstring.com/pages/useragentstring.php?name=Chrome

Apps | S | EC Live | De Anza | BB | Other bookmarks

User Agent String.Com

Home | List of User Agent Strings | Links | API | Contact

Chrome User Agent Strings

Chrome

Free open-source web browser developed by Google. Chromium is the name of the open source project behind Google Chrome, released under the BSD license.

Click on any string to get more details

Chrome 41.0.2228.0

- Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36

Chrome 41.0.2227.1

- Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2227.1 Safari/537.36

Chrome 41.0.2227.0

- Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2227.0 Safari/537.36
- Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2227.0 Safari/537.36

Demo Program: makequery2.py

```
import urllib.request
import urllib.parse
from contextlib import closing

values = {'q':"HTTP"}    # dictionary format
data = urllib.parse.urlencode(values)
cgi = "https://www.google.com/search"
url = cgi + "?" + data

request = urllib.request.Request(url)  # make request to google.com
request.add_header('User-Agent', "Mozilla/5.0 Chrome/41.0.2227.0 Safari/537.36")
response = urllib.request.urlopen(request)              # receive response from google.com
response_data = response.read()

print(response_data.decode('utf8'))
response.close()
```

add_header

Mozilla

Chrome

Safari

Standards for web APIs: Post Methods

LECTURE 1

URL Encoding: **usrlib.parse.urlencode()**

- Sometimes you want to send **data** to a **URL** (often the URL will refer to a **CGI** (Common Gateway Interface) script or other web application).
- With **HTTP**, this is often done using what's known as a **POST** request. This is often what your browser does when you submit a **HTML** form that you filled in on the web.
- Not all **POST**s have to come from forms: you can use a **POST** to transmit arbitrary data to your own application.
- In the common case of **HTML** forms, the **data** needs to be encoded in a standard way, and then passed to the Request object as the data argument. The encoding is done using a function from the **urllib.parse** library.

usrlib.parse.urlencode()

```
import urllib.parse
import urllib.request
url = 'http://www.someserver.com/cgi-bin/register.cgi’
values = {'name' : 'Michael Foord’,
          'location' : 'Northampton’,
          'language' : 'Python’
         }
data = urllib.parse.urlencode(values)
req = urllib.request.Request(url, data)
response = urllib.request.urlopen(req)
the_page = response.read()
```

GET Method

If you do not pass the data argument, **urllib** uses a **GET** request.

One way in which **GET** and **POST** requests differ is that **POST** requests often have “side-effects”: they change the state of the system in some way (for example by placing an order with the website for a hundredweight of tinned spam to be delivered to your door).

Though the **HTTP** standard makes it clear that **POST**s are intended to always cause side-effects, and **GET** requests never to cause side-effects, nothing prevents a **GET** request from having side-effects, nor a **POST** requests from having no side-effects.

Data can also be passed in an **HTTP GET** request by encoding it in the **URL** itself.

GET Method Example

```
import urllib.parse
import urllib.request
data = {}
data['name'] = 'Somebody Here'
data['location'] = 'Northampton'
data['language'] = 'Python'
url_values = urllib.parse.urlencode(data)
url = 'http://www.example.com/example.cgi'
full_url = url + '?' + url_values
data = urllib.request.open(full_url)
```

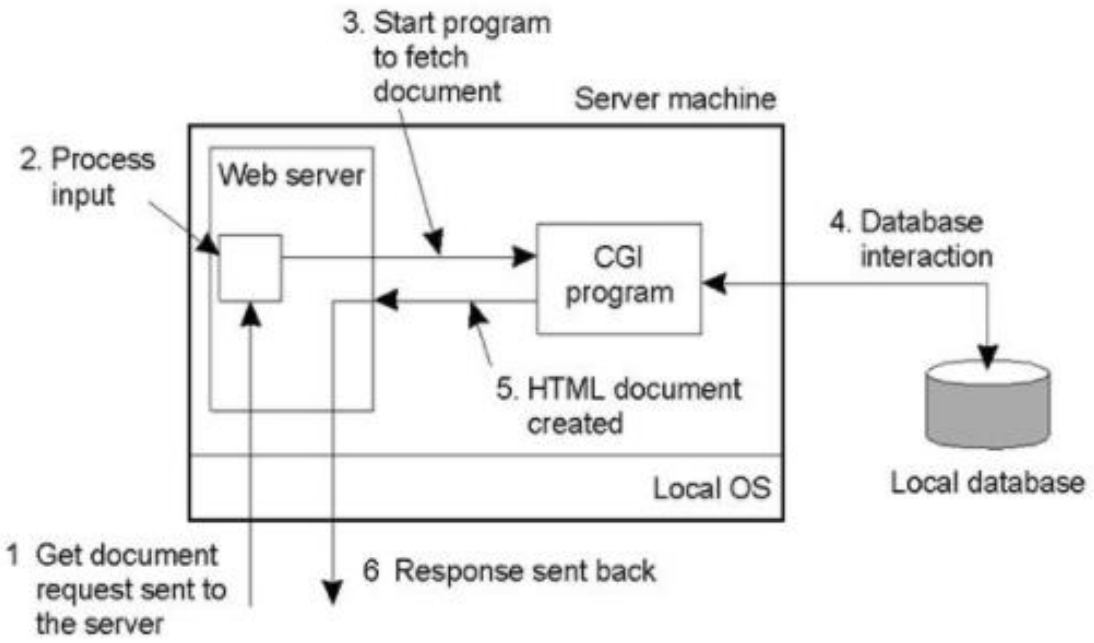
CGI: Common Gateway Interface

An **HTTP server** is often used as a gateway to a legacy information system; for example, an existing body of documents or an existing database application.

The Common Gateway Interface is an agreement between **HTTP** server implementors about how to integrate such gateway scripts and programs.

It is typically used in conjunction with **HTML** forms to build database applications.

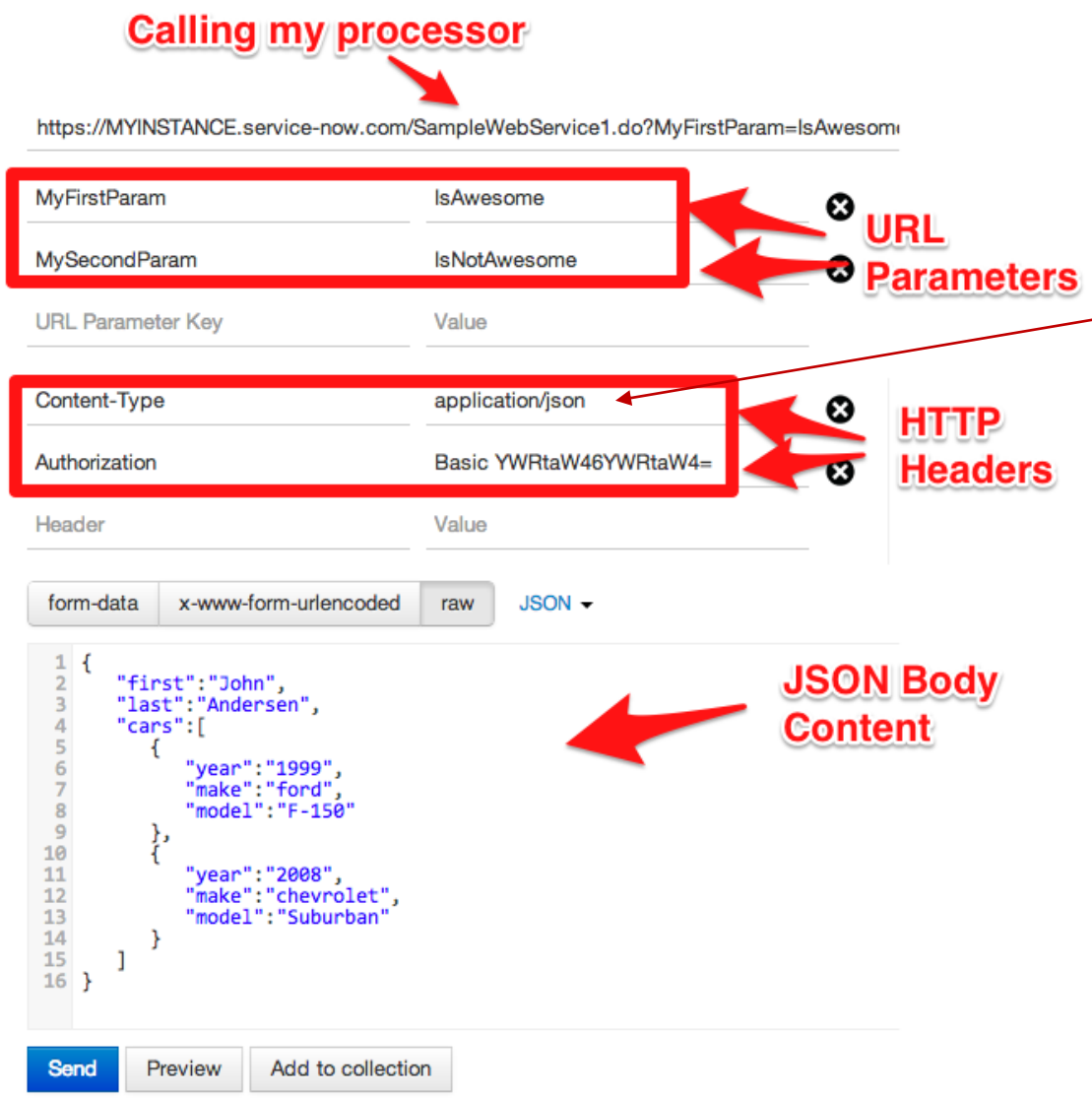
Common Gateway Interface (CGI)



- The principle of using server-side CGI programs.
  - Allows documents can be generated dynamically "on-the-fly"
  - Provides a standard way for web server to execute a program using user-provided data as input
  - To the server, CGI program appears as program responsible for fetching the requested document

Standards for web APIs: Response Data

LECTURE 1

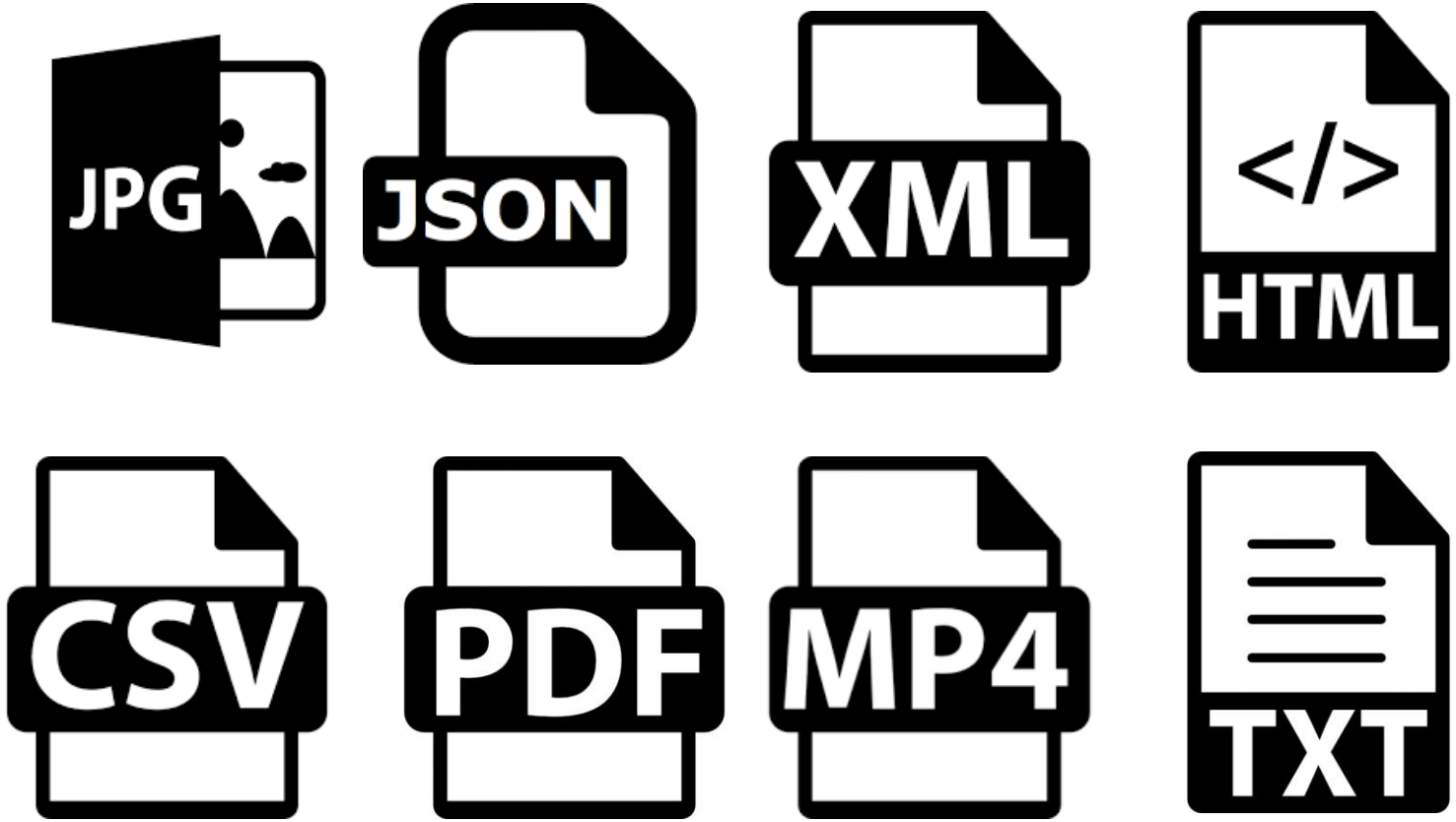


Some other common ContentType values:

```
<%response.ContentType="text/HTML"%>
<%response.ContentType="image/GIF"%>
<%response.ContentType="image/JPEG"%>
<%response.ContentType="text/plain"%>
```

HTML: text/html, full-stop.
XHTML: application/xhtml+xml
XML: text/xml, application/xml (RFC 2376).

Content-Type:



JSON (Javascript Object Notation)

LECTURE 1

JSON

- JavaScript Object Notation
- Minimal
- Textual
- Subset of JavaScript
- Fast conversion to other programming languages.

JSON	Python
object	dict
array	list
string	unicode
number (int)	int, long
number (real)	float
TRUE	TRUE
FALSE	FALSE
null	None

JSON

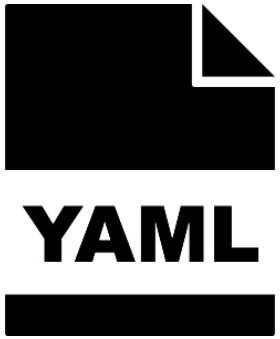
- A Subset of ECMA-262 Third Edition.
- Language Independent.
- Text-based.
- Light-weight.
- Easy to parse.

History of Data Formats

- Ad Hoc
- Database Model
- Document Model
- Programming Language Model

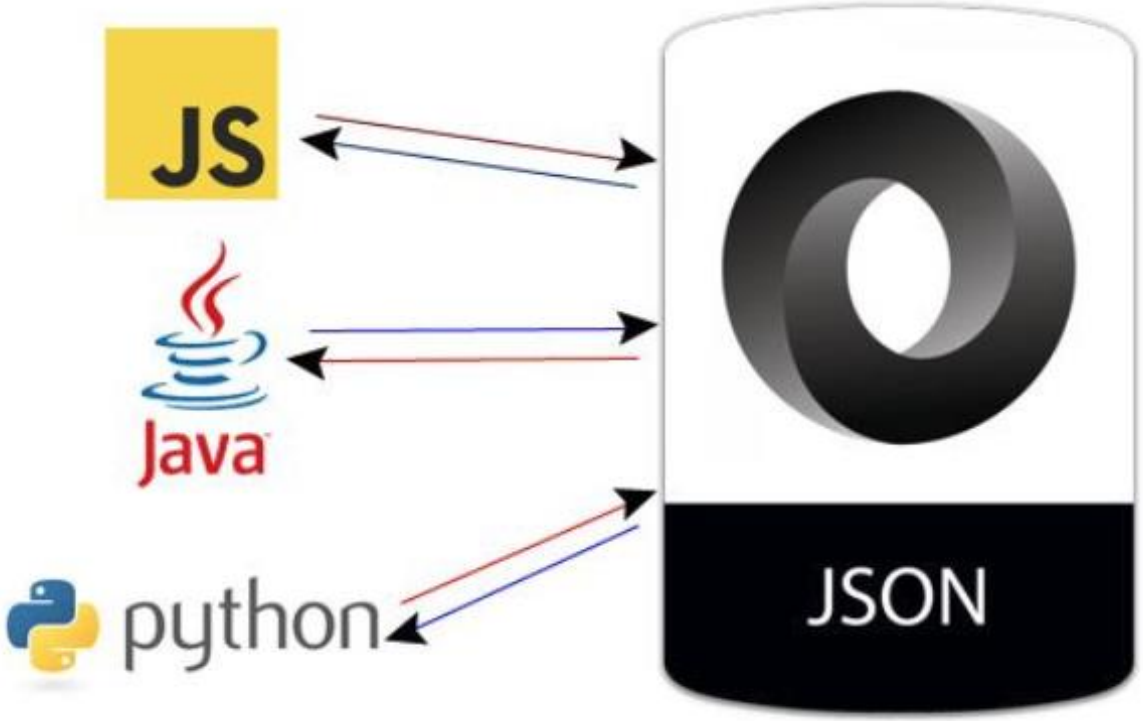
Markup Language

Text/json:
Dictionary in String



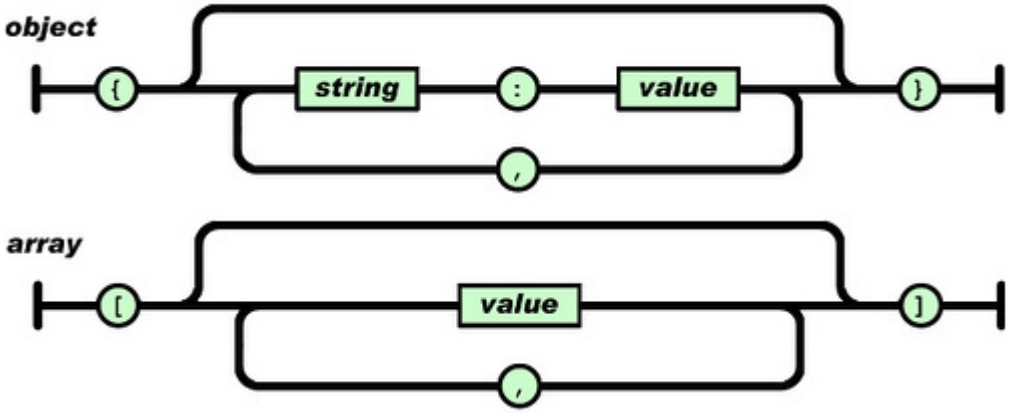
Languages

ActionScript
C / C++
C#
Cold Fusion
Delphi
E
Erlang
Java
Lisp
Perl
Objective-C
Objective CAML
PHP
Python
Rebol
Ruby
Scheme
Squeak



JSON Format

JSON uses name/value pairs. It also has a number of basic data types including numbers, strings, booleans, and null. It also supports **arrays [1, 2]** (like python list) and **objects ("Name":"Tome")** (like python dictionary)




```
{
    hey: "guy",
    anumber: 243,
  - anobject: {
        whoa: "nuts",
      - anarray: [
            1,
            2,
            "thr<h1>ee"
        ],
        more: "stuff"
    },
    awesome: true,
    bogus: false,
    meaning: null,
    japanese: "明日がある。",
    link: http://jsonview.com,
    notLink: "http://jsonview.com is great"
}
```

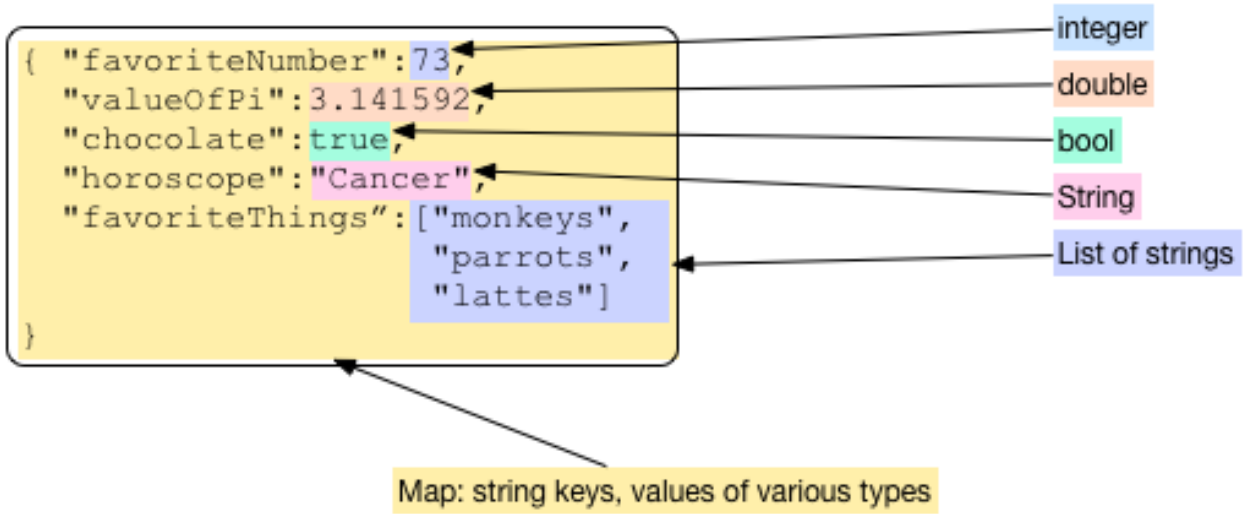
JSON’s Name/Value Pairs

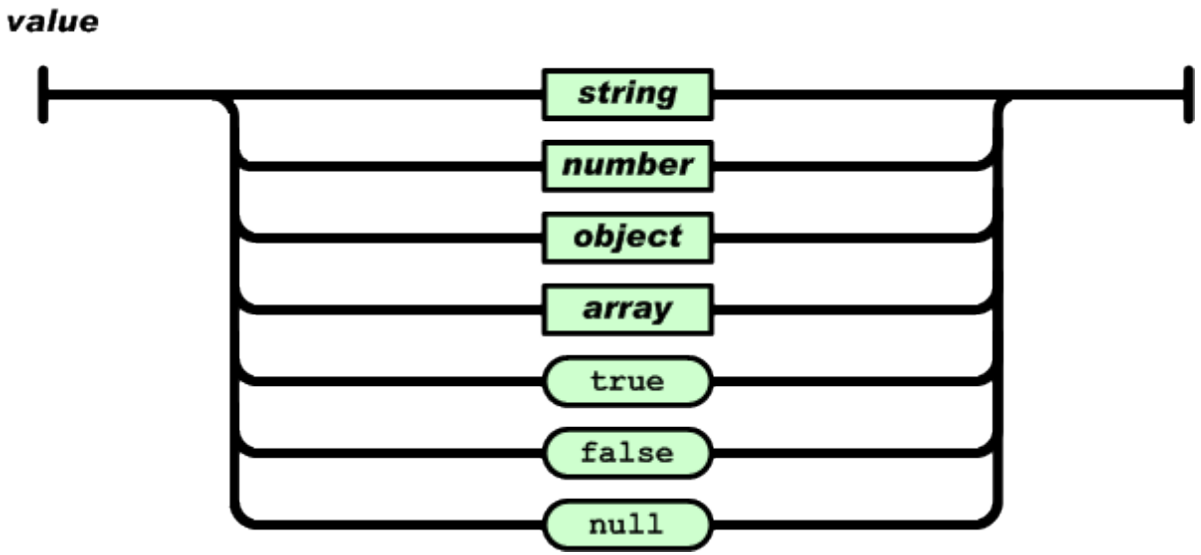
JSON’s name/value pairs are collected in a structured object bounded by curly brackets. Arrays are indicated by square brackets.

JSON’s syntax matches **JavaScript**, but typically a parse function is used to convert JSON text to a JavaScript object. This adds a level of protection from malicious code since JSON data is often sent over the Internet on an unsecure channel. It also addresses bad data issues.

JSON is often used with **JavaScript Ajax** techniques to exchange data. This can provide a more dynamic, interactive interface for a Web page. JSON support is found in most Web browsers.

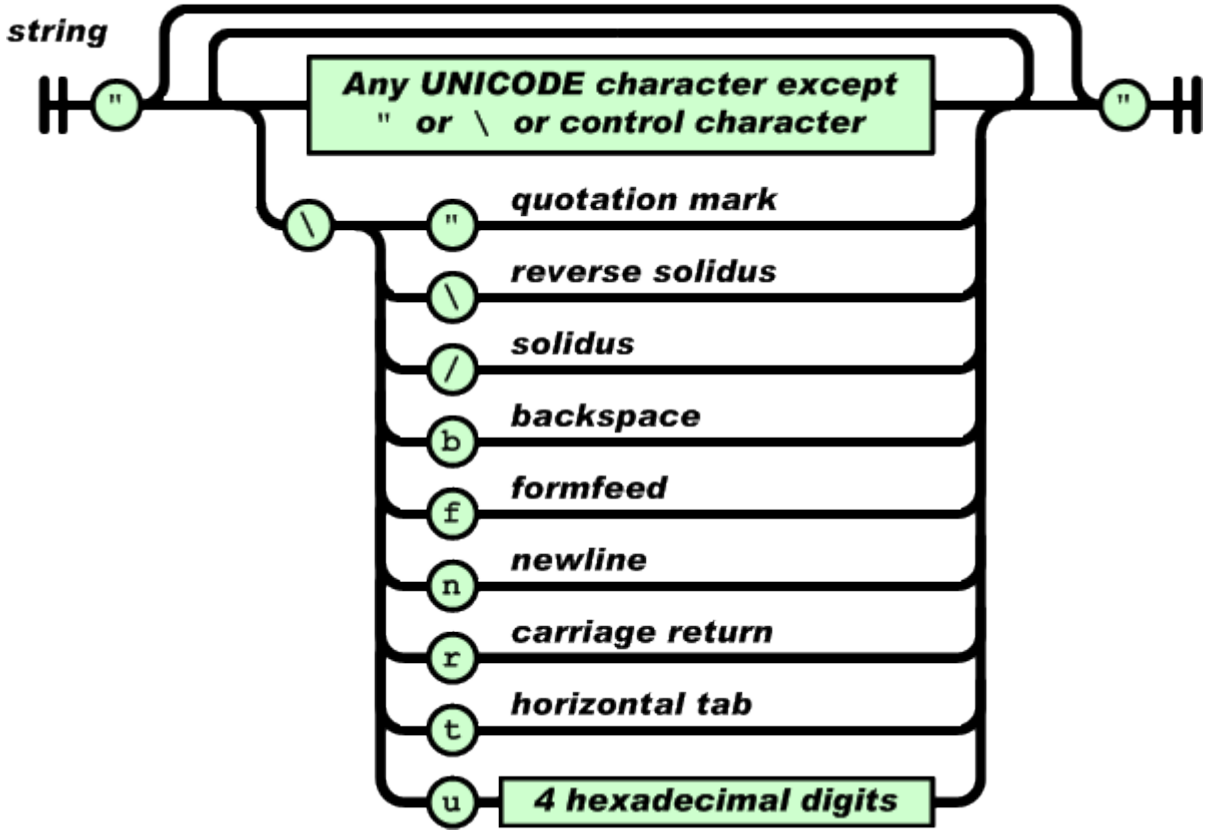
Values


{ "favoriteNumber":73,
"valueOfPi":3.141592,
"chocolate":true,
"horoscope":"Cancer",
"favoriteThings":["monkeys",
"parrots",
"lattes"]
}
integer
double
bool
String
List of strings
Map: string keys, values of various types


value
string
number
object
array
true
false
null

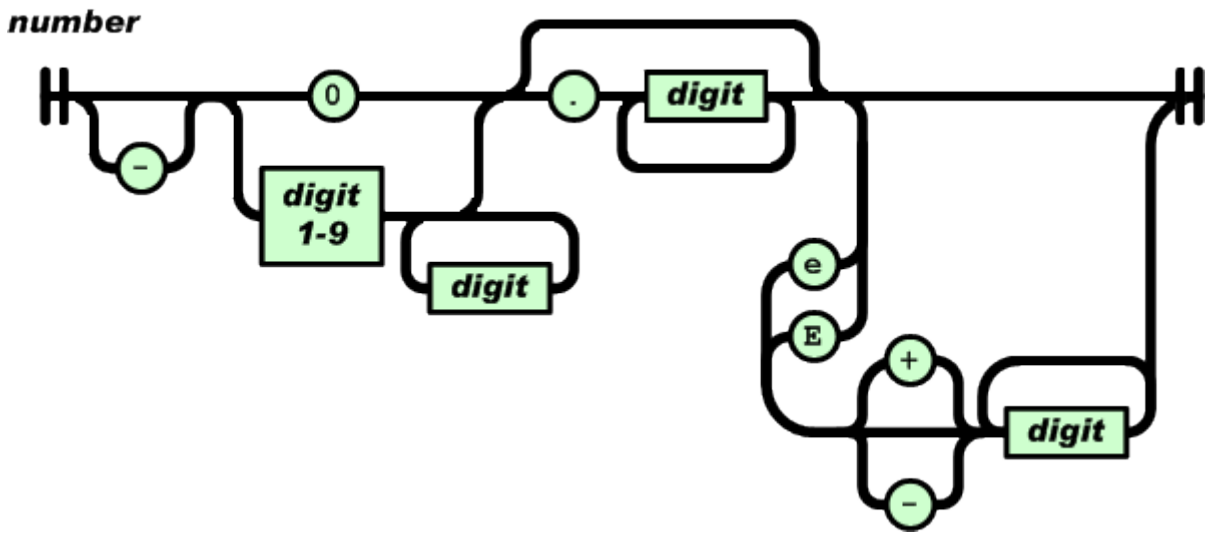
Strings

- Sequence of 0 or more Unicode characters
- No separate character type
  - A character is represented as a string with a length of 1
- Wrapped in "double quotes"
- Backslash escapement



Numbers

- Integer
- Real
- Scientific
- No octal or hex
- No **`NaN`** or **`Infinity`**
  - Use **`null`** instead



Booleans

`true`

`false`

`null`

A value that isn't anything

Object

- Objects are unordered containers of key/value pairs
- Objects are wrapped in **{ }**
  - **,** separates key/value pairs
  - **:** separates keys and values
- Keys are strings
- Values are JSON values
  - struct, record, hashtable, object

Arrays vs Objects

- Use objects when the key names are arbitrary strings.
- Use arrays when the key names are sequential integers.
- Don't get confused by the term Associative Array.

MIME Media Type

application/json

Character Encoding:

- Strictly UNICODE.
- Default: UTF-8.
- UTF-16 and UTF-32 are allowed.

Versionless:

- JSON has no version number.
- No revisions to the JSON grammar are anticipated.
- JSON is very stable.

Rules

- A JSON decoder must accept all well-formed JSON text.
- A JSON decoder may also accept non-JSON text.
- A JSON encoder must only produce well-formed JSON text.
- Be conservative in what you do, be liberal in what you accept from others.

Supersets

- YAML is a superset of JSON.
  - A YAML decoder is a JSON decoder.
- JavaScript is a superset of JSON.
  - A JavaScript compiler is a JSON decoder.
- New programming languages based on JSON.
- Python 3 enforce the bytes-type for data to be transmitted.

Parsing JSON

- Take the following string containing JSON data:

```
json_string = '{"first_name": "Guido", "last_name":"Rossum"}'
```

- It can be parsed like this:

```
import json
parsed_json = json.loads(json_string)
```

- and can now be used as a normal dictionary:

```
print(parsed_json['first_name'])
"Guido"
```

Demo Program: json1.py

Go PyCharm!!!

```
# json1.py: convert from json string to a dictionray
import json
json_string = "{\"first_name\": \"Guido\", \"last_name\":\"Rossum\"}"

parsed_json = json.loads(json_string)
print(parsed_json['first_name'])
```

Run json1

```
C:\Python\Python36\python.exe
Guido
```

Demo Program: json2.py

Go PyCharm!!!

```python
# convert from Python dictionary to Json string
import json
d = {
    'first_name': 'Guido',
    'second_name': 'Rossum',
    'titles': ['BDFL', 'Developer'],
}
print(json.dumps(d))
```

Run json2

```
C:\Python\Python36\python.exe "C:/Eric_Chou/Python Course/Python Object-Oriented Programming
{"first_name": "Guido", "second_name": "Rossum", "titles": ["BDFL", "Developer"]}
```

Youtube Data API

LECTURE 1

YouTube Data API

- The **YouTube Data API** allows us to send a wide variety of different kinds of requests, but we'll focus on just one for this example. Our goal is to issue a search query — like we might do on YouTube's web page — and display the titles and descriptions of videos that match the request.
- The appropriate request in the YouTube Data API is called search, which is described in detail here:
  https://developers.google.com/youtube/v3/docs/search/list

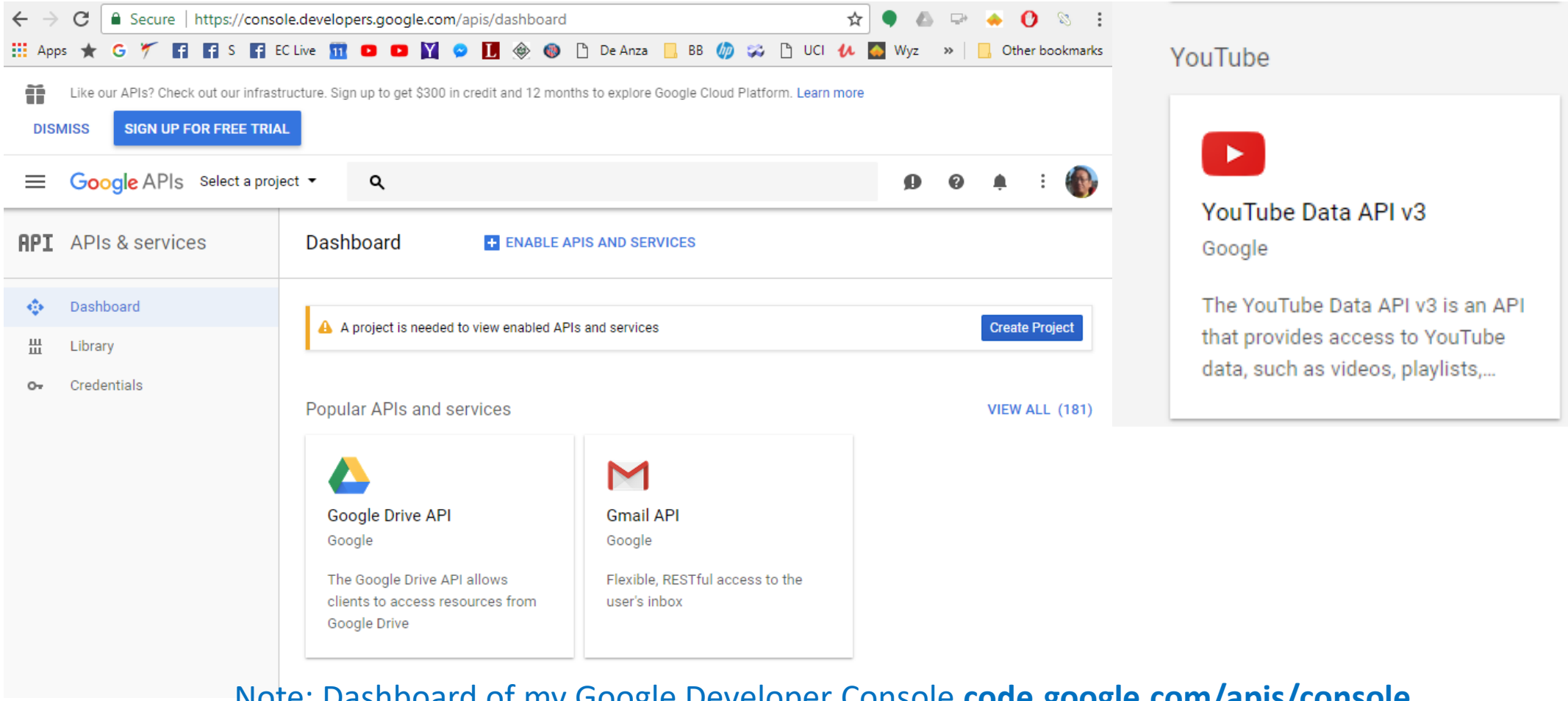
CGI: https://www.googleapis.com/youtube/v3/search

Following the word search in the URL, we add a ? character and then include a list of query parameters. There are lots of parameters we might like to pass, but these are the ones we need in order to accomplish our goal:

- **key**, which is an API key that uniquely identifies us as a user of Google's web APIs. We're required to set up an API key and associate with our Google account if we want to use Google's APIs, though I've already done that for us (and I'll send out an API key in an email message). You can feel free to create your own, too, if you'd like, by logging into your Google account and then visiting the Google Developers Console.
- **type**, which specifies what we want to search for. In our case, we want to search for video, though we could also search for other things (channel or playlist) instead.
- **part**, which specifies what part of YouTube's information you're interested in seeing. In our case, we want something called a snippet, which briefly describes a few aspects of each video.
- **maxResults**, which specifies how many results we want (at most).
- **q**, which specifies our search query.

How to get an API key?

http://help.dimsemenov.com/kb/wordpress-royalslider-tutorials/wp-how-to-get-youtube-api-key



Note: Dashboard of my Google Developer Console **code.google.com/apis/console**

Create Credential and get an API Key

Like our APIs? Check out our infrastructure. Sign up to get $300 in credit and 12 months to explore Google Cloud Platform. Learn more

Google APIs Test

API APIs & services

Dashboard

Library

Credentials

Credentials

Credentials OAuth consent screen Domain verification

Create credentials Delete

Create credentials to access your enabled APIs. Refer to the API documentation for details.

API keys

Name	Creation date	Restrictions	Key
API key 1	Nov 15, 2017	None	AIzaSyBhfd8gXgkfwHO9Gs8cPEa2Cw9Jc3av2MU

Demo Program: youtube.py

Go PyCharm!!!

```python
import urllib.request
import urllib.parse
import codecs

MY_YOUTUBE_API_KEY = "AIzaSyBhfd8gXgkfwHO9Gs8cPEa2Cw9Jc3av2MU"
# prepare API request
data = {}
data['key']  = MY_YOUTUBE_API_KEY
data['type'] = 'video'
data['part'] = 'snippet'
data['maxResults'] = '10'
data['q'] = 'lakers clippers'
data_query = urllib.parse.urlencode(data)
url = "https://www.googleapis.com/youtube/v3/search?"
url_query = url+data_query

# headers needed to pass the web-site auto-bot check.
headers = {}
headers['User-Agent'] = "Mozilla/5.0 (X11; Linux i686)"  # server type at google

# make API request
request = urllib.request.Request(url_query, headers=headers) # make request to google.com
response = urllib.request.urlopen(request)    # just like open a file
response_data = response.read()
response_text = response_data.decode("utf8")  # convert bytes to utf8

# convert text to a html file
lines = response_text.splitlines()
fp = codecs.open("lakers.json", "w", "utf8")    # open a file for utf-8 text format
for line in lines:
    fp.write(line+"\n")
fp.close()
response.close()
```

Youtube Project Key

API Parameters

CGI with ? Question mark
URL for Query

Header Setting to Bypass
Autobot Check

Make Request

Save in lakers.json

Screenshot of Partial Results from **lakers.json** file


```
1  {
2    "kind": "youtube#searchListResponse",
3    "etag": "\"ld9biNPKjAjgjV7EZ4EKeEGrhao/ptNS9pVijaP-6nlR4XCVnzS2YuQ\"",
4    "nextPageToken": "CAoQAA",
5    "regionCode": "US",
6    "pageInfo": {
7     "totalResults": 1000000,
8     "resultsPerPage": 10
9    },
10   "items": [
11    {
12     "kind": "youtube#searchResult",
13     "etag": "\"ld9biNPKjAjgjV7EZ4EKeEGrhao/xkt6r53_TFZo-GAPTDVapexkK_g\"",
14     "id": {
15      "kind": "youtube#video",
16      "videoId": "5GWbpRvYR6Q"
17     },
```

Demo Program: youtube2.py

Go PyCharm!!!

The response message (JSON string) is converted into python dictionary.

```python
import urllib.request
import urllib.parse
import codecs
import json

MY_YOUTUBE_API_KEY = "AIzaSyBhfd8gXgkfwHO9Gs8cPEa2Cw9Jc3av2MU"
# prepare API request
data = {}
data['key']  = MY_YOUTUBE_API_KEY
data['type'] = 'video'
data['part'] = 'snippet'
data['maxResults'] = '10'
data['q'] = 'lakers clippers'
data_query = urllib.parse.urlencode(data)
url = "https://www.googleapis.com/youtube/v3/search?"
url_query = url+data_query

# headers needed to pass the web-site auto-bot check.
headers = {}
headers['User-Agent'] = "Mozilla/5.0 (X11; Linux i686)"  # server type at google

# make API request
request = urllib.request.Request(url_query, headers=headers) # make request to google.com
response = urllib.request.urlopen(request)   # just like open a file
response_data = response.read()
response_text = response_data.decode("utf8")  # convert bytes to utf8

# load dictionary
d = json.loads(response_text)
print("kind:"+d['kind'])
print("etag:"+d['etag'])
keylist = list(d.keys())
count = 0
for key in keylist:
    print(key, end=" ")
    if count % 20==19:
        print()
    count = count + 1
response.close()
```

Load the json string to python dictionary

Parsing JSON Objects

LECTURE 1

Response Message

- The response message may contains HTML file, an XML file or a JSON file.
- These files can be in compressed text format.
- In order to used these files, we need to learn how to parse these files properly.
- For HTML and XML files, we can use typical text editor or HTML editor to understand them.
- In this lecture, we focus on JSON files.

To Parse JSON Response Message

- We need to have the description for the JSON message from the Server's API.
- We may save the JSON message to a text file.

```
# make API request
request = urllib.request.Request(url_query, headers=headers) # make request to google.com
response = urllib.request.urlopen(request)    # just like open a file
response_data = response.read()
response_text = response_data.decode("utf8")   # convert bytes to utf8

# convert text to a html file
lines = response_text.splitlines()
fp = codecs.open("lakers.json", "w", "utf8")    # open a file for utf-8 text format
for line in lines:
    fp.write(line+"\n")
fp.close()
response.close()
```

Make Request

Save in lakers.json

View JSON file by JSON Editor

Data Types:

- Number (integer)
- Number (real)
- Boolean (True/False)
- null

Data Structures:

- Array: [1, 2, 3]
- Object: { “Name”: “Eric Chou”,
  “Score”: 98,
  }

JSON	Python
object	dict
array	list
string	unicode
number (int)	int, long
number (real)	float
TRUE	TRUE
FALSE	FALSE
null	None

JSON Editor Online

New | Open ▾ | Save ▾ | Settings ▾ | Help

powered by ace


```
1  {
2   "kind": "youtube#searchListResponse",
3   "etag": "\"ld9biNPKjAjgjV7EZ4EKeEGrhao
      /BVS8kybQbhtkDXAXt3Ly5akme9o\"",
4   "nextPageToken": "CAoQAA",
5   "regionCode": "US",
6   "pageInfo": {
7    "totalResults": 1000000,
8    "resultsPerPage": 10
9   },
10  "items": [
11   {
12    "kind": "youtube#searchResult",
13    "etag": "\"ld9biNPKjAjgjV7EZ4EKeEGrhao/xkt6r53_TFZo
        -GAPTDVapexkK_g\"",
14    "id": {
15     "kind": "youtube#video",
16     "videoId": "5GWbpRvYR6Q"
17    },
18    "snippet": {
19     "publishedAt": "2017-10-20T05:09:42.000Z",
20     "channelId": "UCoh_z6QB0AGB1oxWufvbDUg",
21     "title": "Los Angeles Lakers vs LA Clippers Full Game
        Highlights / Week 1 / 2017 NBA Season",
22     "description": "Los Angeles Lakers vs LA Clippers Full Game
        Highlights / Week 1 / 2017 NBA Season For more information,
        as well as all the latest NBA news and highlights, ...",
23     "thumbnails": {
24      "default": {
25       "url": "https://i.ytimg.com/vi/5GWbpRvYR6Q/default.jpg",
26       "width": 120,
27       "height": 90
28      },
29      "medium": {
30       "url": "https://i.ytimg.com/vi/5GWbpRvYR6Q/mqdefault.jpg",
31       "width": 320,
```


Ln: 1 Col: 1

object ▶ items ▶

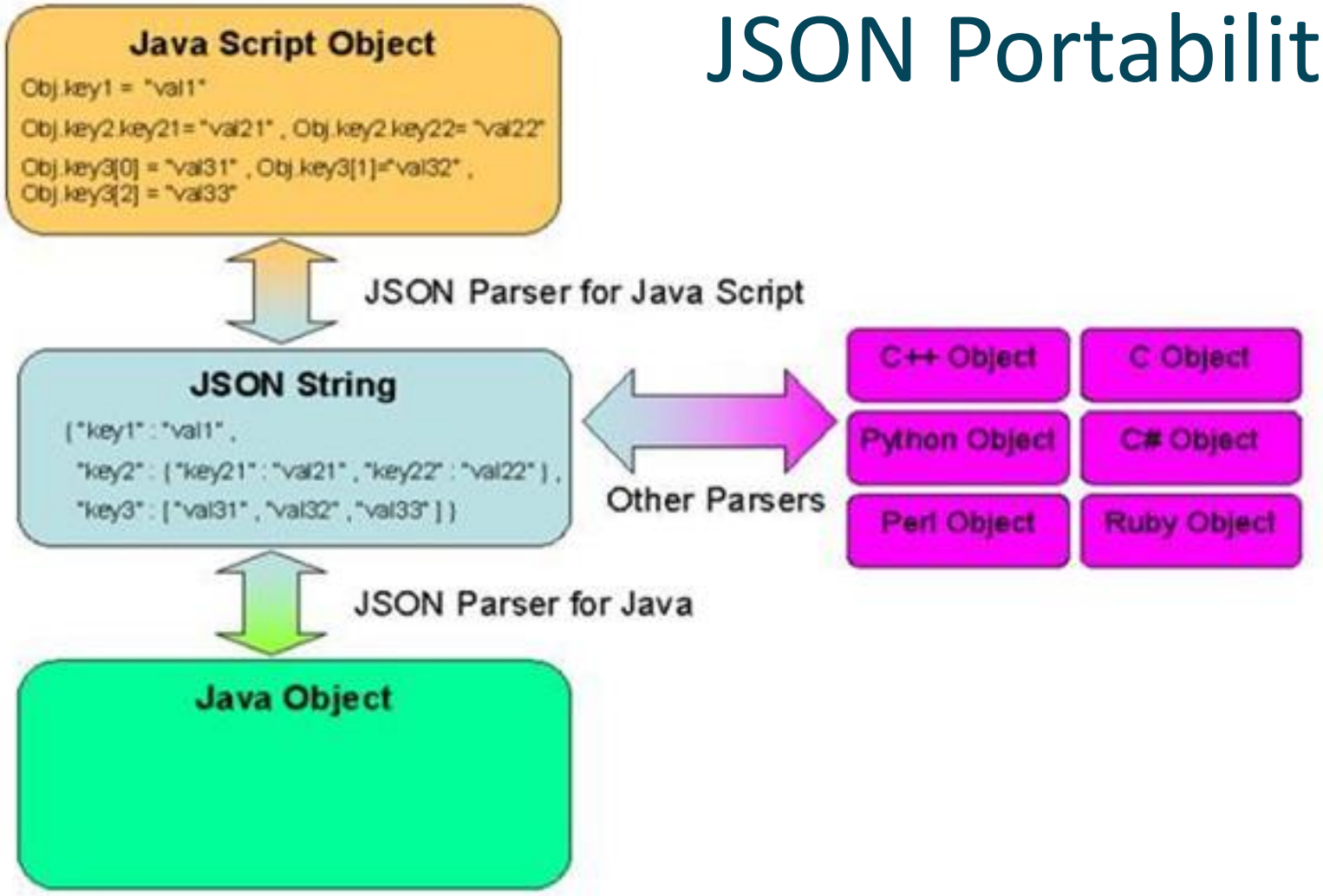
```
▼ object {6}
    kind : youtube#searchListResponse
    etag : \"ld9biNPKjAjgjV7EZ4EKeEGrhao/BVS8kybQbhtkDXAXt3Ly5akme
           9o\"
    nextPageToken : CAoQAA
    regionCode : US
  ▼ pageInfo {2}
      totalResults : 1000000
      resultsPerPage : 10
  ▼ items [10]
    ▶ 0 {4}
    ▶ 1 {4}
    ▶ 2 {4}
    ▶ 3 {4}
    ▶ 4 {4}
    ▶ 5 {4}
    ▶ 6 {4}
    ▶ 7 {4}
    ▶ 8 {4}
    ▶ 9 {4}
```

JSON arrys (Python list)

JSON Object (Python Dictionary)

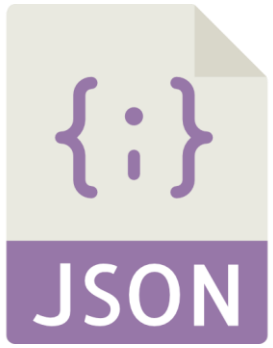
laker.json

JSON Portability



Python Parser and Encoder

import json

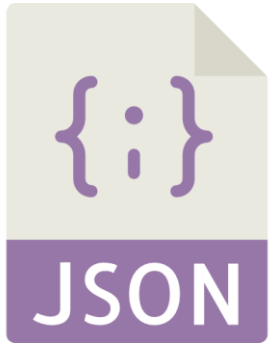



```
fp = open("a.json", "r")
str = fp.read()
```

Python String:
"{"key": "value"}"

Python JSON Parser:
dictionary = json.loads(str)

Python dictionary




```
fp = open("b.json", "w")
fp.write(str)
```

Python String:
"{"key": "value"}"

Python JSON Encoder:
str = json.dumps(['foo', {dictionary)

Python JSON Encoders

json.dump(obj, fp, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)

- Serialize **obj** as a JSON formatted stream to fp (a .write()-supporting file-like object) using this conversion table.

json.dumps(obj, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)

- Serialize **obj** to a JSON formatted str using this conversion table. The arguments have the same meaning as in dump().

Python JSON Parsers

json.load(fp, *, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)

- Deserialize **fp** (a .read()-supporting file-like object containing a JSON document) to a Python object using this conversion table.

json.loads(s, *, encoding=None, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)

- Deserialize **s** (a str, bytes or bytearray instance containing a JSON document) to a Python object using this conversion table.

Demo Program: person.json, person.py

Go JSON Editor On-line!!!

Then, Go PyCharm!!!

```python
import json

fp = open("person.json", "r")
str = fp.read()
person = json.loads(str)

print("Name=", person['name'])
print("Score=", person['score'])

friends = person['friends']
for i in range(len(person['friends'])):
    print("Friend[%d]=%s" % (i, friends[i]))

person['name'] = "Mr."+person['name']
person['score'] = person['score'] -5

str_out = json.dumps(person)
fout = open("person_out.json", "w")
fout.write(str_out)

fp.close()
fout.close()
```