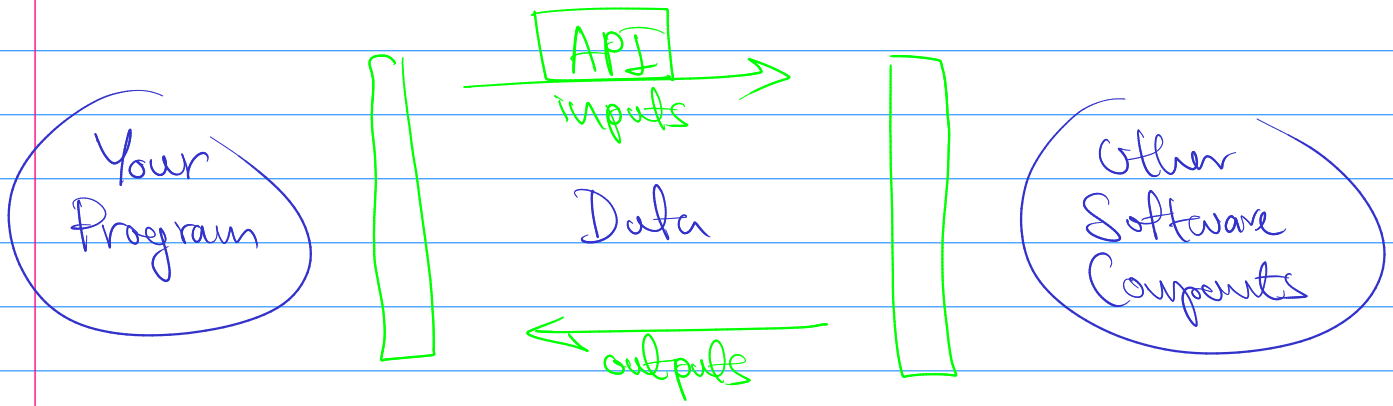
Week 5

APIs, and Data Collection

video 1 : Simple APIs

Application Program Interface

what is API



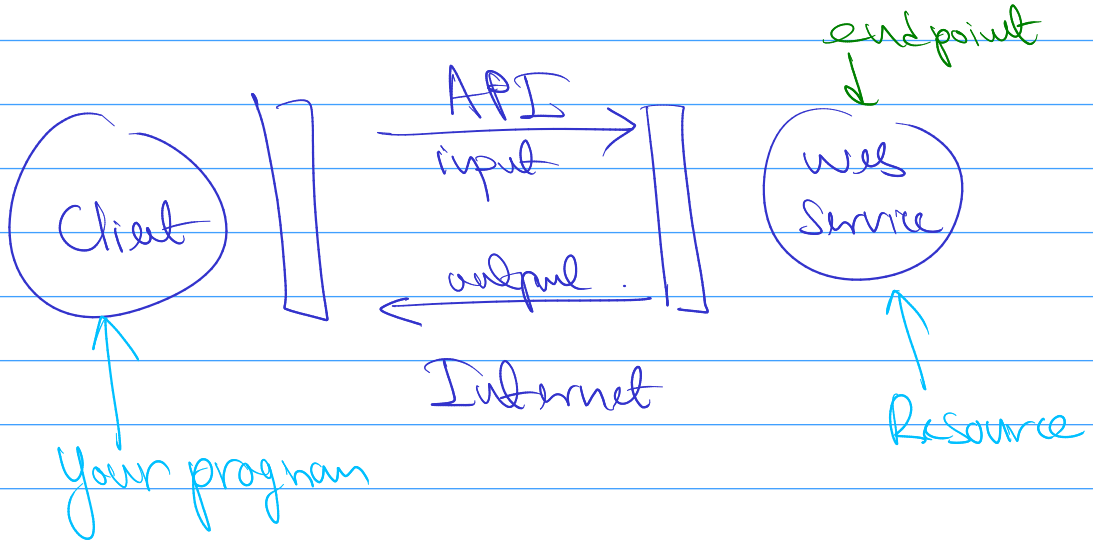
example API → Pandas Object

```
df = pd.DataFram({'a':[...], 'b':[
```

df.head() ← Pandas API gives result for this

REST APIs

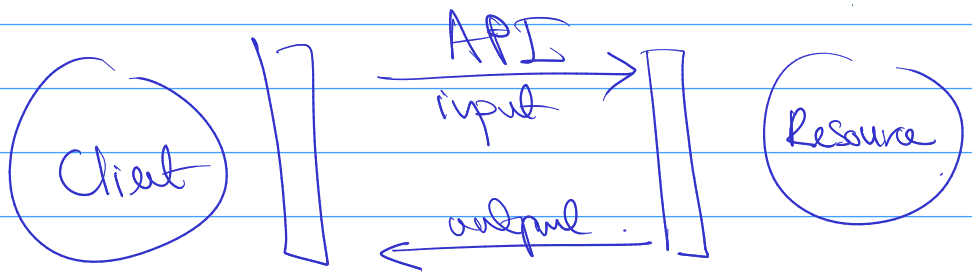
REpresentational State Transfer APIs



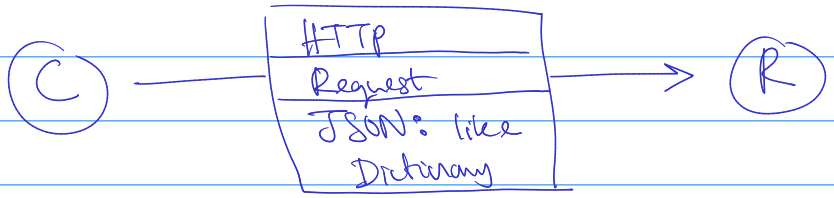
Set of Rules regarding
- Communication
- Input or Request
- Output or Response

REST API over HTTP

↑ way of transferring data

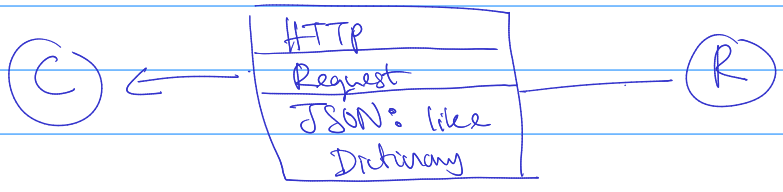


(1) Client sends a request to the Resource as HTTP massage



(2) Web service proform the operation

(3) Resource response as HTTP massage



exaple from Crypto Curaney Data

This is perfect exaple becuses
Constatly updated
Vital to CryptoCurreny Trading

we use PyCoinGecko Python Client/wrapper
for the CoinGecko API

we also use pandas time series futtion
to deal with time series data.

```
!pip install pycoingecko
from pycoingecko import CoinGeckoAPI
cg = CoinGeckoAPI()
bitcoin_data = cg.get_coin_market_chart_by_id(id='bitcoin', vs_currency='usd', days=30)
```

function to request

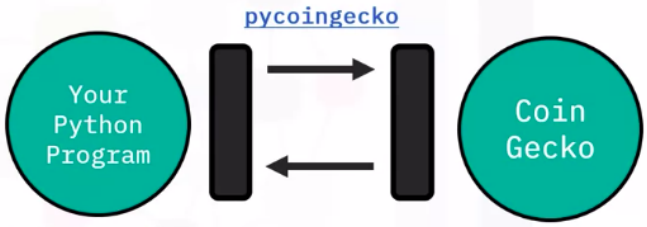
client objet.

past 30 days

response JSON file

python dictionary

```
bitcoin_data:
{'prices':
[[1604138559526,
13927.73252295053],
[1604142031697,
13797.785093120927],
[1604145789426,
13917.92602548896],
[1604150065202,
13873.651848313166],
[1604152981403,
13845.28388774076],
[1604156700430,
13806.349989300157],…
```




```
bitcoin_data['prices']:
[[1604138559526, 13927.73252295053],
[1604142031697, 13797.785093120927],
[1604145789426, 13917.92602548896],
[1604150065202, 13873.651848313166],
[1604152981403, 13845.28388774076]]
```

IBM Developer SKILLS NETWORK

price, market cap
total volume
unic time stamps

we select only price

dictionary → into → dataframe

columns

```
data = pd.DataFrame(bitcoin_price_data, columns=['TimeStamp', 'Price'])
```

	TimeStamp	Price
0	1604138559526	13927.732523
1	1604142031697	13797.785093
2	1604145789426	13917.926025
3	1604150065202	13873.651848
4	1604152981403	13845.283888
...	...	...
716	1606716898166	18510.032786
717	1606720068286	18509.742863
718	1606724609861	18439.969378
719	1606727338318	18453.345378
720	1606728336000	18465.880043

Pandas fn
to_datetime

Date
2020-10-31 10:02:39.526
2020-10-31 11:00:31.697
2020-10-31 12:03:09.426
2020-10-31 13:14:25.202
2020-10-31 14:03:01.403
...
2020-11-30 06:14:58.166
2020-11-30 07:07:48.286
2020-11-30 08:23:29.861
2020-11-30 09:08:58.318
2020-11-30 09:25:36.000

Simplify Dates

IBM Developer 4:25 / 5:12 SKILLS NETWORK

```
data['Date'] = pd.to_datetime(data['TimeStamp'], unit='ms')
```

	TimeStamp	Price	Date
0	1604138559526	13927.732523	2020-10-31 10:02:39.526
1	1604142031697	13797.785093	2020-10-31 11:00:31.697
2	1604145789426	13917.926025	2020-10-31 12:03:09.426
3	1604150065202	13873.651848	2020-10-31 13:14:25.202
4	1604152981403	13845.283888	2020-10-31 14:03:01.403
...	...	...	...
716	1606716898166	18510.032786	2020-11-30 06:14:58.166
717	1606720068286	18509.742863	2020-11-30 07:07:48.286
718	1606724609861	18439.969378	2020-11-30 08:23:29.861
719	1606727338318	18453.345378	2020-11-30 09:08:58.318
720	1606728336000	18465.880043	2020-11-30 09:25:36.000

IBM Developer 4:41 / 5:12 SKILLS NETWORK

Candle Stick Plots

we need min, max, ____ for the plot

```
candlestick_data = data.groupby(data.Date.dt.date).agg({'Price': ['min', 'max',
'first', 'last']})
```

	Price			
	min	max	first	last
Date				
2020-10-31	13720.425685	13917.926025	13797.785093	13796.028786
2020-11-01	13661.927858	13822.470734	13778.637638	13671.267179
2020-11-02	13313.920358	13786.247252	13786.247252	13554.676070
2020-11-03	13359.343536	13847.027174	13558.361796	13847.027174
2020-11-04	13588.251443	14122.045525	13973.977815	14120.620632
2020-11-05	14095.428885	15511.856787	14170.702323	15511.856787
2020-11-06	15413.862067	15855.288073	15570.112495	15619.662967
2020-11-07	14630.212278	15679.761910	15583.677358	14926.198182
2020-11-08	14749.879849	15561.599966	14749.879849	15506.766777

Plot

```
fig = go.Figure(data=[go.Candlestick(x= candlestick_data.index,
                open=candlestick_data['Price']['first'],
                high=candlestick_data['Price']['max'],
                low=candlestick_data['Price']['min'],
                close=candlestick_data['Price']['last'])
                ])

fig.update_layout(xaxis_rangeslider_visible=False, xaxis_title='Date',
yaxis_title='Price (USD $)', title='Bitcoin Candlestick Chart Over Past 30 Days' )


plot(fig, filename='bitcoin_candlestick_graph.html')
```


Bitcoin Candlestick Chart Over Past 30 Days
Price (USD $)
19k
18k
17k
16k
15k
14k
13k
Nov 1
2020
Nov 8
Nov 15
Nov 22
Nov 29
Date
Play
IBM Developer
5:08 / 5:12
SKILLS NETWORK

Video 2

APIs that use some kind of AI

Lab: audio file → English text → Spanish text

API keys and Endpoints

API key: way to access API
: Unique Characters
: API identifies you

Lab: ① Sign up for a API key

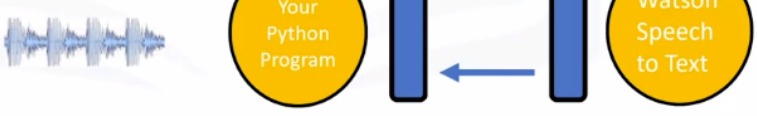
② download an audio file

```
from ibm_watson import SpeechToTextV1

url_s2t="https://stream.watsonplatform.net/speech-to-text/api"
iam_apikey_s2t="EOeiZxxxDxV2xxxxxxxxxxxxxxxxxjYen9SKARKW-"
s2t = SpeechToTextV1(iam_apikey=iam_apikey_s2t, url=url_s2t)
```

speech to text adapter object

endpoint

API key



771 772 937
Vgi982

downloaded audio

```
filename='hello_this_is_python.wav'
with open(filename, mode="rb")  as wav:
    response = s2t.recognize(audio=wav, content_type='audio/wav')
```

read in binary format

object

file format

method recognize for speech to text V1

this sends audio file to watson

IBM Developer SKILLS NETWORK

Then Watson sends a response

```
response.result
```

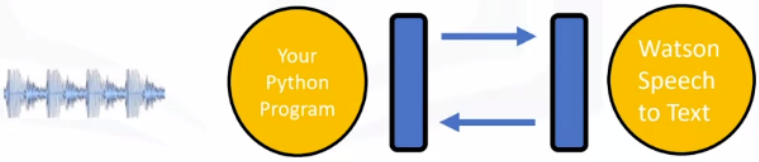
```
{'results': [{'alternatives': [{'confidence': 0.91, 'transcript' 'hello this is
python '}], 'final': True}], 'result_index': 0}
```

```
recognized_text=response.result['results'][0]["alternatives"][0]["transcript"]
```

Select only the transcript

```
recognized_text:
'hello this is python '
```

final result.



IBM Developer SKILLS NETWORK

Next part English text → Spanish text

```
from ibm_watson import LanguageTranslatorV3

url_lt = 'https://gateway.watsonplatform.net/language-translator/api'

apikey_lt = 'dU2SaxxxxxxxxxxxxxxxasdfCuasdf'

version_lt = '2018-05-01'

language_translator = LanguageTranslatorV3(iam_apikey=apikey_lt, url=url_lt,version=version_lt)

language_translator.list_identifiable_languages().get_result()
```

```
{'languages': [{'language': 'af', 'name': 'Afrikaans'}, {'language': 'ar', 'name': 'Arabic'}, {'language': 'az', 'name':
'Azerbaijani'}, {'language': 'ba', 'name': 'Bashkir'}, {'language': 'be', 'name': 'Belarusian'}, {'language': 'bg', 'name':
'Bulgarian'}, {'language': 'bn', 'name': 'Bengali'}, {'language': 'bs', 'name': 'Bosnian'}, {'language': 'ca', 'name':
'Catalan'}, {'language': 'cs', 'name': 'Czech'}, {'language': 'cv', 'name': 'Chuvash'}, {'language': 'da', 'name': 'Danish'},
{'language': 'de', 'name': 'German'}, {'language': 'el', 'name': 'Greek'}, {'language': 'en', 'name': 'English'}, {'language':
'eo', 'name': 'Esperanto'}, {'language': 'es', 'name': 'Spanish'}
```

IBM Developer SKILLS NETWORK

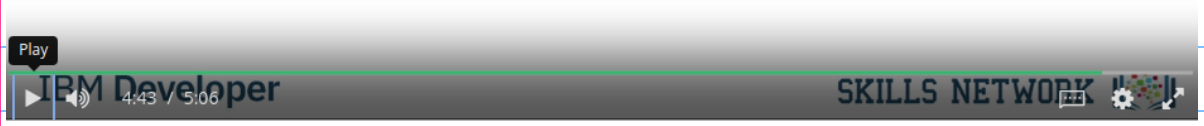
```
recognized_text = 'hello this is python '

translation_response = language_translator.translate(text=recognized_text, model_id='en-es')

translation =translation_response.get_result()
```

English → Spanish

```
translation
{'translations': [{'translation': 'Hola, esta es la pitón. '}],
'word_count': 4, 'character_count': 21}

spanish_translation =translation['translations'][0]['translation']
spanish_translation
'Hola, esta es la pitón. '
```

← isolate translation

← result.



```
translation_new = language_translator.translate(text=spanish_translation ,
model_id='es-en').get_result()

translation_eng=translation_new['translations'][0]['translation']

translation_eng
'Hey, this is the python. '
```

Translate back to English

```
French_translation=language_translator.translate(text= translation_eng,
model_id='en-fr').get_result()

French_translation['translations'][0]['translation']
"Hé, c'est le python. "
```

Translate to French

IBM Developer SKILLS NETWORK

Language Translator IBM Watson Service

YptfoUuKzUakVOAlbbbVLzvpwjUEzQgb_JWUrdq77dOF

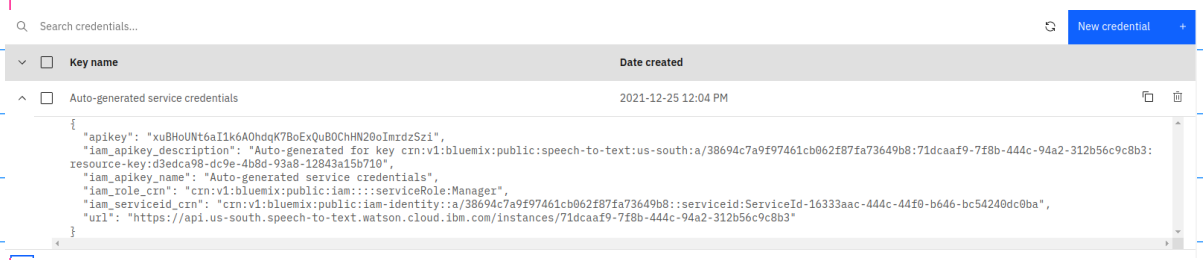
https://api.us-south.language-translator.watson.cloud.ibm.com/instances/33cd26b1-89ef-4cf5-a670-a6692f1308a4

Speech to Text IBM Watson Services


```
{
  "apikey": "xuBHoUNt6al1k6AOhdqK7BoExQuBOChHN20olmrdzSzi",
  "iam_apikey_description": "Auto-generated for key crn:v1:bluemix:public:speech-to-text:us-south:a/38694c7a9f97461cb062f87fa73649b8:71dcaaf9-7
  "iam_apikey_name": "Auto-generated service credentials",
  "iam_role_crn": "crn:v1:bluemix:public:iam::::serviceRole:Manager",
  "iam_serviceid_crn": "crn:v1:bluemix:public:iam-identity::a/38694c7a9f97461cb062f87fa73649b8::serviceid:ServiceId-16333aac-444c-44f0-b646-bc
  "url": "https://api.us-south.speech-to-text.watson.cloud.ibm.com/instances/71dcaaf9-7f8b-444c-94a2-312b56c9c8b3"
}
```


Saw


```
{
  "apikey": "xuBHoUNt6aI1k6AOhdqK7BoExQuBOChHN20oImrdzSzi",
  "iam_apikey_description": "Auto-generated for key crn:v1:bluemix:public:speech-to-text:us-south:a/38694c7a9f97461cb062f87fa73649b8:71dcaaf9-7f8b-444c-94a2-312b56c9c8b3:resource-key:d3edca98-dc9e-4b8d-93a8-12843a15b710",
  "iam_apikey_name": "Auto-generated service credentials",
  "iam_role_crn": "crn:v1:bluemix:public:iam::::serviceRole:Manager",
  "iam_serviceid_crn": "crn:v1:bluemix:public:iam-identity::a/38694c7a9f97461cb062f87fa73649b8::serviceid:ServiceId-16333aac-444c-44f0-b646-bc54240dc0ba",
  "url": "https://api.us-south.speech-to-text.watson.cloud.ibm.com/instances/71dcaaf9-7f8b-444c-94a2-312b56c9c8b3"
}
```

Second Part of Week 5

REST APIs & HTTP Requests

HTTP : Hypar Text Trawfer Prafocal

HTTP : General Prafocall trawferiy information throw thn web

HTTP massage usually contans a JSON file

Uniform Resource Locator : URL

pgaular way to find resources in web

3 parts of URL

for this lab

① Scheme: the protocal (http://, ...)

② Internet address or Base URL : used to find thn location (www.ibm.com, www.gitlab.com, ...)

③ Route : location on thn web server (/images/IDSNlogo.png, ...)

Request Massage

Request Message for the get request method

Request Start line	Get/index.html HTTP/1.0
Request Header	User-Agent: python-requests/2.21.0 Accept-Encoding: gzip, deflate :
Request Body	

requested file
get method
additional information

Response Massage

Response Message

Response Start line	HTTP/1.0 200 OK
Response Header	Server: Apache-Cache: UNCACHEABLE
Response Body	<!DOCTYPE html> <html> <body> My First Heading My first paragraph. </body> </html>

version number
diriptive code
Status code
200 → Success
information
requested file
in this case HTML file

Status Code

1XX	Informational
100	Everything So Far Is OK
2xx	Success
200	OK
3XX	Redirection
300	Multiple Choices

4XX	Client Error
401	Unauthorized
403	Forbidden
404	Not Found
500	Server Error
501	No Implemented

HTTP Methods

HTTP METHODS	Description
GET	Retrieves data from the server
POST	Submits data to server
PUT	Updates data already on server
DELETE	Deletes data from server

previous request example

REST APIs & HTTP Requests – Part 2

Request Module in Python
allws send http/1.1 request easily

import requests

Requests

```
import requests
url='https://www.ibm.com/'
r=requests.get(url)
r.status_code:200
r.request.headers
{'User-Agent': 'python-requests/2.24.0', 'Accept-Encoding': 'gzip, deflate',
'Accept': '*/*', 'Connection': 'keep-alive', 'Cookie':
'_abck=A5C90067E0241F8BBD3ECC70ECDB1EC0~-
1~YAAQLc2U0ZaliKB1AQAAwucY4QT77mnX/GJQJvRGOV48PDVR70euYZ9FTBsFbF3z4kwEcKgttV+Ot6Sz
P+sjsPDIFni428WfQn/yUYVddQ2e1ejPilhGVE+eNJl4xbIRXxE1U59jQLvi0jd/Q8e6ClE7mw27Qzb5nC
gdMcd4ZnULKeu6U5QSVLFRUPnhMLvOM40+1WFW1CNH36WrccSUtoKkTsExTltvbsvxIbthin3znfL1racK
tV9WJFJlAEdqajkt0qX/frGBULGLK/r47j78DuZHabdMhdalss9J1MpfXj6kXCFe~-1~-1~-1;
bm_sz=564D34683F98DB203E92B73A05A3324C~YAAQLc2U0ZWliKB1AQAAwucY4Qn7jwZFH9NO7QY5q55
JvdADHcL95RCSPsAGLlx2RAr+iIH6hEt/frpWT8RXkjWBcCtS29lu91pIEz6zjed+o0yyTH7aL/qkC9nzV
mdZfvAhOjFKnFzewR7xguYLBc7XlAAeG8GrkPFHq0vMTo38GwTMCC+xGYQYpM6y'}
```

response object

Get request

status code

see the request headers

IBM Developer SKILLS NETWORK

Requests

```
r.request.body:None
header=r.headers
header:
{'Server': 'Apache', 'x-drupal-dynamic-cache': 'UNCACHEABLE', 'Link':
'<https://www.ibm.com/ca-en>; rel="canonical", <https://www.ibm.com/ca-en>; rel="revision",
<https://www.ibm.com/ca-en>; rel="revision", <//1.cms.s81c.com>; rel=preconnect; crossorigin,
<//1.cms.s81c.com>; rel=dns-prefetch', 'x-ua-compatible': 'IE=edge', 'Content-Language': 'en-
ca', 'x-generator': 'Drupal 8 (https://www.drupal.org)', 'x-dns-prefetch-control': 'on', 'x-
drupal-cache': 'MISS', 'Last-Modified': 'Thu, 19 Nov 2020 10:32:43 GMT', 'ETag':
'"1605781963"', 'Content-Type': 'text/html; charset=UTF-8', 'x-acquia-host': 'www.ibm.com',
'x-acquia-path': '/ca-en', 'x-acquia-site': '', 'x-acquia-purge-tags': '', 'x-varnish':
'18482279 12683819', 'x-cache-hits': '9', 'x-age': '7030', 'Accept-Ranges': 'bytes',
'Content-Encoding': 'gzip', 'Cache-Control': 'public, max-age=300', 'Expires': 'Thu, 19 Nov
2020 15:26:47 GMT', 'X-Akamai-Transformed': '9 11615 0 pmb=mTOE,1', 'Date': 'Thu, 19 Nov 2020
15:21:47 GMT', 'Content-Length': '11725', 'Connection': 'keep-alive', 'Vary': 'Accept-
Encoding', 'x-content-type-options': 'nosniff', 'X-XSS-Protection': '1; mode=block',
'Content-Security-Policy': 'upgrade-insecure-requests', 'Strict-Transport-Security': 'max-
age=31536000', 'x-ibm-trace': 'www-dipatcher: dynamic rule'}
```

see request body

see response header

This is Not the request header

IBM Developer 1:21 / 4:56 SKILLS NETWORK

Requests

key — date the request send

```
header['date']:'Thu, 19 Nov 2020 15:21:47 GMT'
header['Content-Type']:'text/html; charset=UTF-8'
r.encoding:'UTF-8'
r.text[0:100]:

'<!DOCTYPE html>\n<html lang="en-ca" dir="ltr">\n
<head>\n <meta charset="utf-8" />\n<script>digitalD'
```

type of data

encoding

content first 100 characters

IBM Developer SKILLS NETWORK

Get Request with URL Parameters

Modify results of your query
retriving data from API

Lab: we use httbin.org

GET request

Base URL	Route
httbin.org	/get
httbin.org/get	

After the Get request we have the Query string

Query string ← part of URL

Start of Query	Parameter Name		Value		Parameter Name		Value
?	name	=	Joseph	&	ID	=	123

http://httpbin.org/get? Name=Joseph&ID=123

previous page

Python ex

Create Query string

```
url_get='http://httpbin.org/get'
payload={"name":"Joseph","ID":"123"}
r=requests.get(url_get,params=payload)
r.url:'http://httpbin.org/get?name=Joseph&ID=123'
r.request.body : None
r.status_code : 200
```

url_get='http://httpbin.org/get' ← base URL + /get

payload={"name":"Joseph","ID":"123"} — for query string

print URL → r.url:'http://httpbin.org/get?name=Joseph&ID=123'

request body → r.request.body : None

print status code → r.status_code : 200

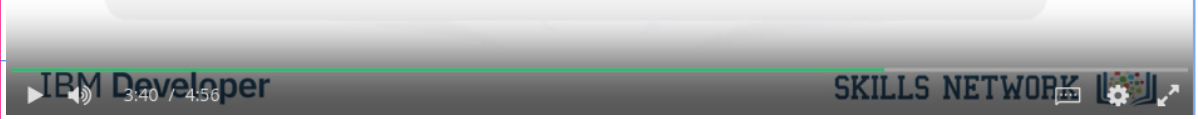
Content-Type

View the response as text

```
r.text
```


```
{ "args": { "ID": "123", "name": "Joseph" },
"headers": { "Accept": "*/*", "Accept-
Encoding": "gzip, deflate", "Host":
"httpbin.org", "User-Agent": "python-
requests/2.24.0", "X-Amzn-Trace-Id": "Root=1-
5fbbd03e-106584ab42513a6d5cef39a9" }, "origin":
"99.228.137.181", "url":
"http://httpbin.org/get?name=Joseph&ID=123" }
```


```
r.headers['Content-Type']   'application/json'
```



look at key content-types to see content type

Content-Type in Json

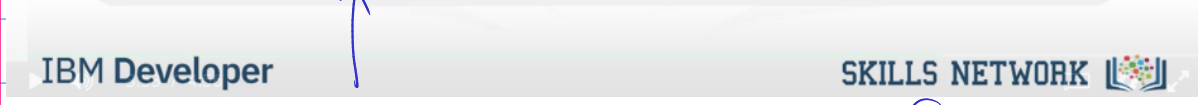
format it using .json()

now it's a dictionary

```
r.json()
```


```
{'args': {'ID': '123', 'name': 'Joseph'},
'headers': {'Accept': '*/*', 'Accept-Encoding':
'gzip, deflate', 'Host': 'httpbin.org', 'User-
Agent': 'python-requests/2.24.0', 'X-Amzn-
Trace-Id': 'Root=1-5fbbd03e-
106584ab42513a6d5cef39a9'}, 'origin':
'99.228.137.181', 'url':
'http://httpbin.org/get?name=Joseph&ID=123'}
```


```
r.json()['args']   {'ID': '123', 'name': 'Joseph'}
```



key 'args' has values for Query String

Post Request : Send data to a server
(Similar to get request)

different

Get → data in URL
Post → data in a request body

POST

```
url_post="http://httpbin.org/post"

payload={"name":"Joseph","ID":"123"}

r_post=requests.post(url_post,data=payload)
```

IBM Developer SKILLS NETWORK

Comparison Get Vs Post

Compare POST and GET

```
print("POST request URL:",r_post.url )
print("GET request URL:",r.url)


POST request URL: http://httpbin.org/post
GET request URL: http://httpbin.org/get?name=Joseph&ID=123
```

no name or value pair in URL

has

IBM Developer SKILLS NETWORK

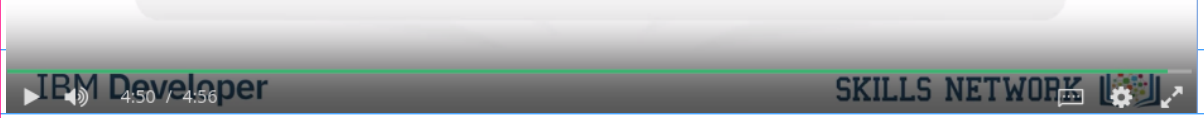
Compare POST and GET

```
print("POST request body:",r_post.request.body)
print("GET request body:",r.request.body)

POST request body: name=Joseph&ID=123
GET request body: None

r_post.json()['form']

{'ID': '123', 'name': 'Joseph'}
```

has

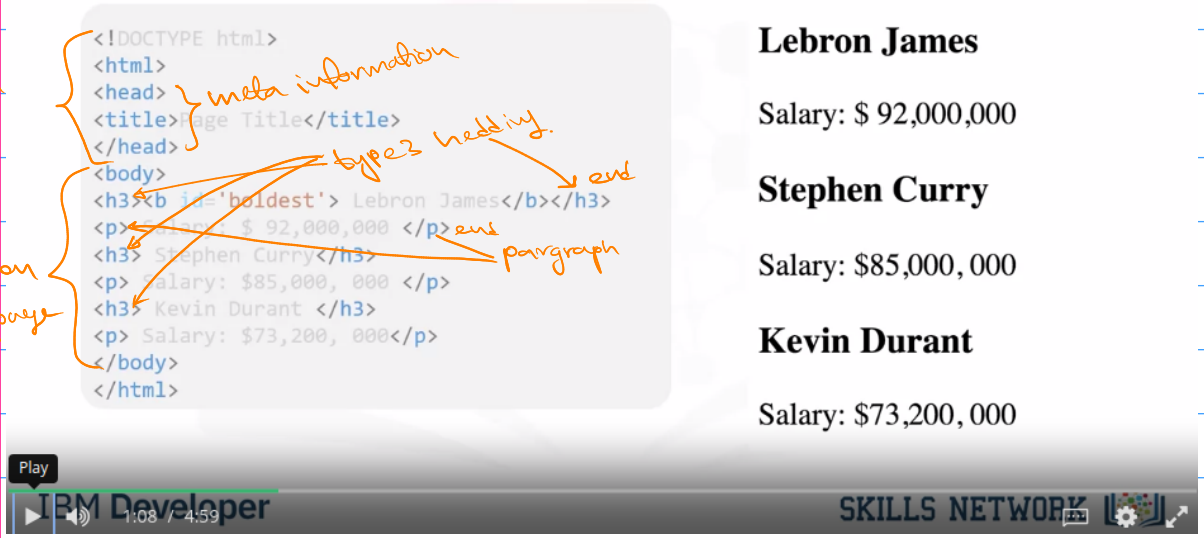
No body

extract payload



Optional : HTML for Web Scraping

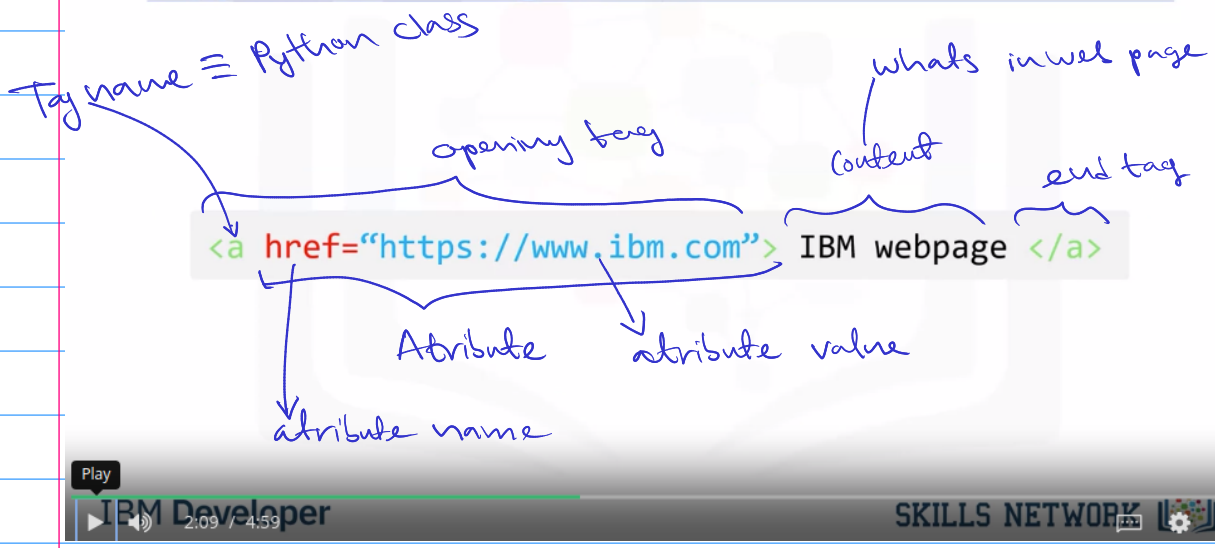
HTML Composition



Composition of an HTML Tag

HTML anchor Tag

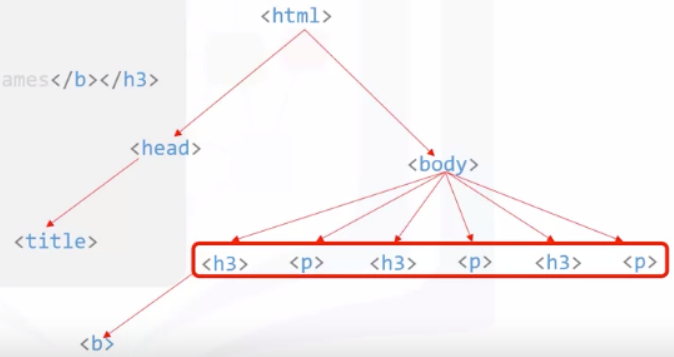
Hyperlink Tag



HTML Tree

Document Tree

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h3>  <b id='boldest'> Lebron James</b></h3>
    <p> Salary: $ 92,000,000 </p>
    <h3> Stephen Curry</h3>
    <p> Salary: $85,000, 000 </p>
    <h3> Kevin Durant </h3>
    <p> Salary: $73,200, 000</p>
   </body>
</html>
```



IBM Developer SKILLS NETWORK

HTML Table

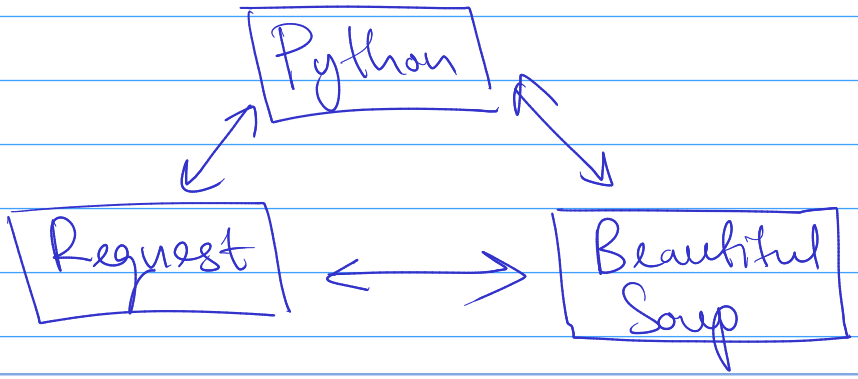
HTML Tables

```
<table >
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
  </tr>
  <tr>
    <td>Domino Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td >144 </td>
  </tr>
</table>
```

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144

IBM Developer 4:39 / 4:59 SKILLS NETWORK

Webscraping



Webscraping example

Lebron James

Salary: $ 92,000,000

Stephen Curry

Salary: $85,000, 000

Kevin Durant

Salary: $73,200, 000

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

IBM Developer 1:06 / 4:59 SKILLS NETWORK

Beautiful Soup

```
from bs4 import BeautifulSoup

html="<!DOCTYPE html><html><head><title>Page
Title</title></head><body><h3><b id='boldest'>Lebron
James</b></h3><p> Salary: $ 92,000,000 </p><h3> Stephen
Curry</h3><p> Salary: $85,000, 000 </p><h3> Kevin Durant </h3><p>
Salary: $73,200, 000</p></body></html>"

soup = BeautifulSoup(html, 'html5lib')
```

web page as a string

next page

Soup object has webpage as a nested data structure.

IBM Developer 1:17 / 4:59 SKILLS NETWORK

When using Beautiful Soup what is the difference between 'lxml' and "html.parser" and "html5lib"?

24

7

When would you use one over the other and the benefits of each? When I used each they seemed to be interchangeable, but people here correct me that I should be using a different one. I'd like to strengthen my understanding; I've read a couple posts on here about this but they're not going over the uses much in any at all.

Example:

```
soup = BeautifulSoup(response.text, 'lxml')
```

python html web-scraping beautifulsoup lxml

Share Improve this question Follow

edited Jan 15 at 2:30
smci
28.3k ● 18 ● 107 ● 142

asked Aug 3 '17 at 21:06
duc hathaway
337 ● 1 ● 2 ● 8

Add a comment

2 Answers

Active | Oldest | Votes

35

From the **docs**'s summarized table of advantages and disadvantages:

1. **html.parser** - `BeautifulSoup(markup, "html.parser")`
   - Advantages: Batteries included, Decent speed, Lenient (as of Python 2.7.3 and 3.2.)
   - Disadvantages: Not very lenient (before Python 2.7.3 or 3.2.2)
2. **lxml** - `BeautifulSoup(markup, "lxml")`
   - Advantages: Very fast, Lenient
   - Disadvantages: External C dependency
3. **html5lib** - `BeautifulSoup(markup, "html5lib")`
   - Advantages: Extremely lenient, Parses pages the same way a web browser does, Creates valid HTML5
   - Disadvantages: Very slow, External Python dependency

Share Improve this answer Follow

answered Aug 3 '17 at 21:26

Vinícius Figueiredo
5,710 ● 3 ● 20 ● 39

Add a comment

Tag Object

tag_object=soup.title

tag_object:
<title>Page Title</title>

tag_object=soup.h3

tag_object:
<h3>Lebron James</h3>

1st <h3>

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

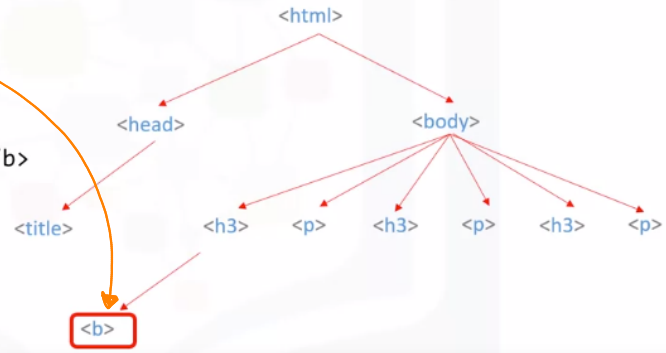
IBM Developer SKILLS NETWORK

HTML Tree

tag_object: <h3>Lebron James</h3>

tag_child =tag_object.b
tag_child:

Lebron James

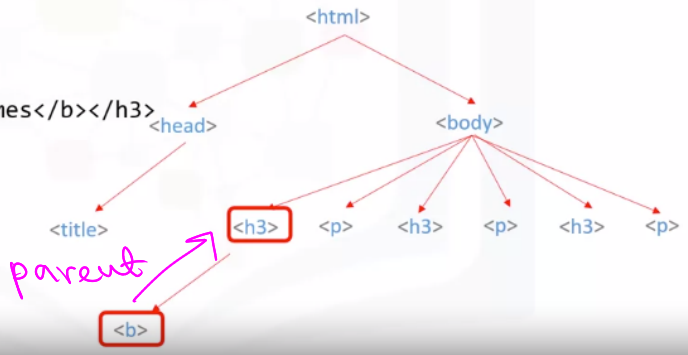


IBM Developer SKILLS NETWORK

.parent

Parent attribute

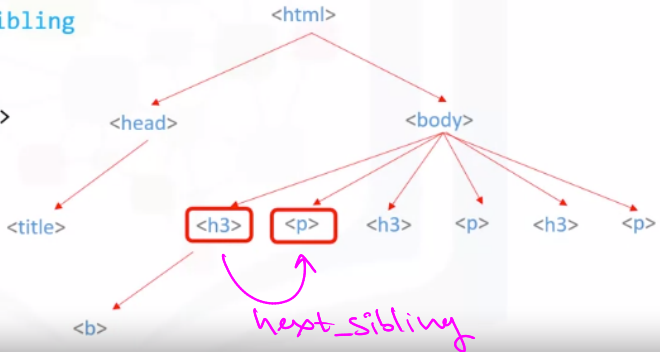
```
tag_child:<b id="boldest">Lebron James</b>

parent_tag=tag_child.parent
parent_tag:

<h3><b id="boldest">Lebron James</b></h3>
```



IBM Developer 2:27 / 4:59 SKILLS NETWORK

Next-sibling attribute

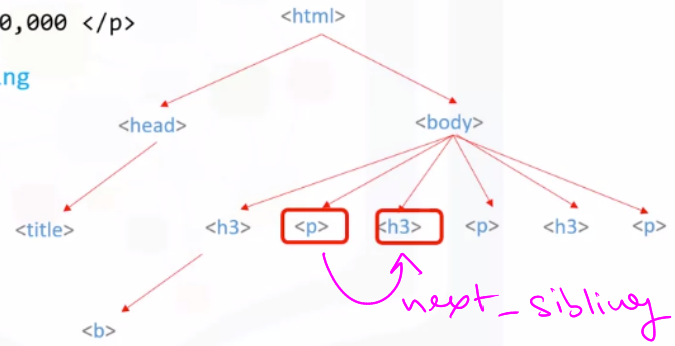
```
tag_object: <h3><b id="boldest">Lebron James</b></h3>

sibling_1= tag_object.next_sibling
sibling_1:

<p> Salary: $ 92,000,000 </p>
```



IBM Developer 2:33 / 4:59 SKILLS NETWORK

Next-sibling attribute

```
sibling_1: <p> Salary: $ 92,000,000 </p>

sibling_2=sibling_1.next_sibling
sibling_2:

<h3> Stephen Curry</h3>
```



IBM Developer SKILLS NETWORK

Navigable string

```
tag_child:<b id="boldest">Lebron James</b>

tag_child.attrs:

{'id': 'boldest'}

tag_child.string:

'Lebron James'
```

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

IBM Developer SKILLS NETWORK

find_all

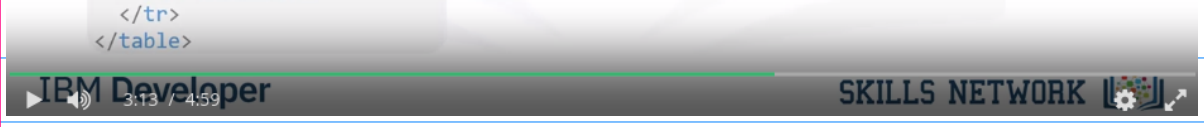
this is a filter, which can be used to filter based on

Tags name
Its atributes
text of a string
or some combination of these.

Pizza place list

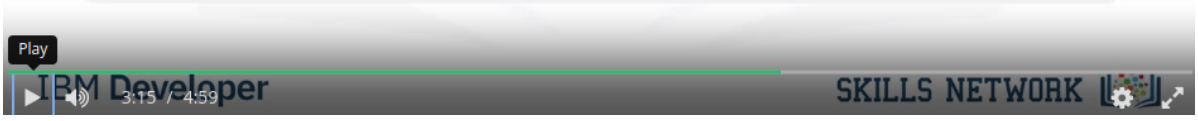
```
<table >
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
  </tr>
  <tr>
    <td>Domino Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td >144 </td>
  </tr>
</table>
```

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144



Beautiful Soup object

```
from bs4 import BeautifulSoup

html= "<table><tr><td>Pizza Place</td><td>Orders</td>
<td>Slices </td></tr><tr><td>Domino'sPizza</td>
<td>10</td><td>100</td></tr><tr><td>Little
Caesars</td><td>12</td><td>144</td></table>"

table = BeautifulSoup(html, 'html5lib')
```

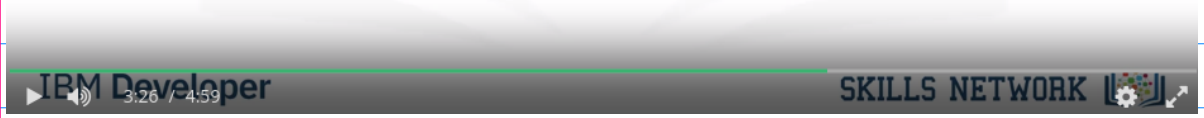


Python iterable

```
table_row=table.find_all(name='tr')


table_row:
```

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144

```
[<tr><td>Pizza Place</td><td>Orders</td><td>Slices</td></tr>,
 <tr><td>Domino's Pizza</td><td>10</td><td>100</td></tr>,
 <tr><td>Little Caesars</td><td>12</td><td>144</td></tr>,
```

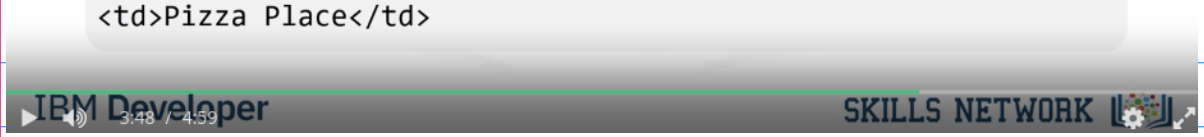


Tag object

```
first_row =table_row[0]
first_row:

<tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr>


first_row.td :

<td>Pizza Place</td>
```

Tag object

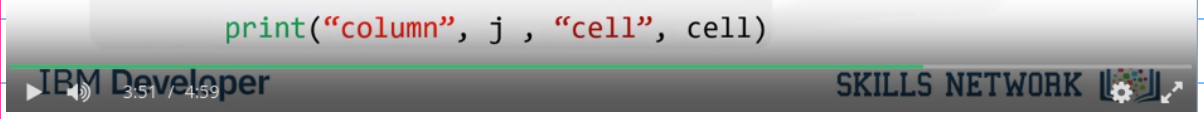
```
first_row =table_row[0]
first_row:

<tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr>


first_row.td :


<td>Pizza Place</td>
```



Variable row

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144
Papa John's	15	166

```
for i,row in enumerate(table_rows):

    print("row", i)
    cells=row.find_all("td")

    for j,cell in enumerate(cells):

        print("column", j , "cell", cell)
```



A Webpage Example

Overview Python Program

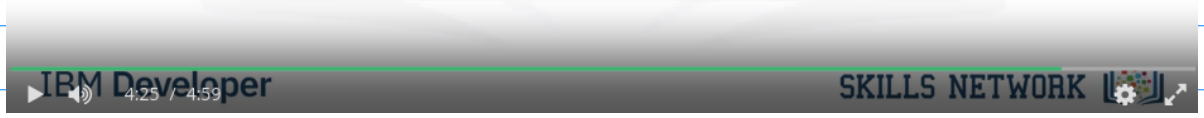
```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text

#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")

# Pulls all instances of <a> tag
artists = soup.find_all('a')

#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

URL

get the text

get method to download webpage

create a beautifulsoup object.



Working with different file formats

csv , xml , json , xlsx

Understanding File Formats

.csv
.json ← file type

library make this easy

```
import pandas as pd

file = "FileExample.csv"

df = pd.read_csv(file)
```

default → first line is header

```
df.columns = ["Name", "Phone Number", "Bday"]
```

Read JSON Files

```
import json

with open('filesample.json', 'r') as openfile:

    json_object = json.load(openfile)

print(json_object)
```

Reading XML File

Extensible Markup Language

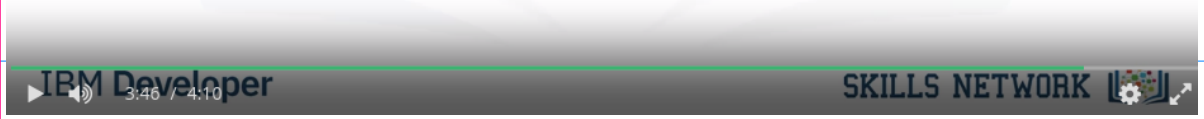
panda doesn't have a attribute to read these files

Reading XML Files

```
import pandas as pd
import xml.etree.ElementTree as etree
tree = etree.parse("fileExample.xml")
root = tree.getroot()
columns = ["Name", "Phone Number", "Birthday"]
df = pd.DataFrame(columns = columns)
```

import this library

for the loop



Reading XML Files

```
for node in root:
    name = node.find("name").text
    phonenumber = node.find("phonenumber").text
    birthday = node.find("birthday").text

df = df.append(pd.Series([name, phonenumber,
birthday], index = columns)
…..., ignore_index = True)
```

go throw the file and put them in dataframe