2. Python for Data Science, AI & Development

Semana 1

Python commands

```
import sys
print(sys.version) #python version
sys.float_info #sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=3
08, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53,
 epsilon=2.220446049250313e-16, radix=2, rounds=1)

type(12) #returns "int"

float(2) #casts 2 from int to float
int('1')
int(True) #returns 1

7//2 #double slashes stand for integer division, it rounds the result. This returns 3
```

Stand out by sharing your notebooks

Data Scientists, Stand out by Sharing Your Notebooks

Posted on December 3, 2018 by Antonio Cangiano Data Science has been dubbed as the sexiest profession of the 21st century. Demand is at an all-time high. Salaries are - particularly when Machine and Deep Learning are thrown into the mix - well into the six figures.

https://cognitiveclass.ai/blog/data-scientists-stand-out-by-sharing-your-notebooks/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2021-01-01

Strings

- negative indexes: Name[-1] its the last char
- Name[::2] → we access every second item
- Name[0:5:2] → every second item from 0 to five (five not included)
- len("asd") → length
- concatenate with +
- 3 * "Michael" → "Michael Michael Michael"
- escape sequences
  - \n → new line
  - \t → tab
  - \\ → \

String methods

- A.upper()
- A.replace('Michael', 'Janet')
- Name.find('el') → returns first index of the sequence. If the sqeuence doesnt exists it returns -1

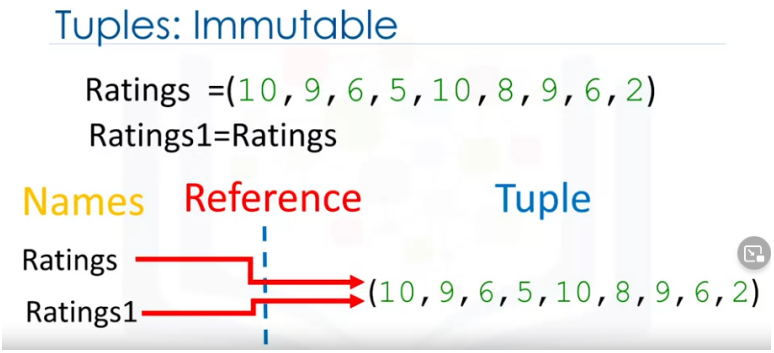
Semana 2

List and tuples

Tuples

- ordered sequence
- tup = (10, 0, 9, 1)

- tup = ("disco", 1, 2.3) → tup[0] = "disco". tup[-3] = "disco". tup[-1] = 2.3
- elements can be access via index
- tup2 = tup1 + ("hard rock", 10) = ("disco", 1, 2.3, "hard rock", 10)
- slice: tup[0:3] (the last index is one larger than the index you want)
- len()
- immutables



- we cant change element 2 because tuples are immutable, we need to create a new one
- sorted()
- NT = (1, 2, ("pop", "rock"), (3, 4), "hola")
- NT[2] = ("pop", "rock") [1] = "rock"

Lists

- ordered sequence
- nesting like tuples
- indexing
- slicing
- concatenate by adding
- L.extend(newList) → this modifies L
- mutable
- L.append([2,3,4]) → append only adds ONE element to the list, in this case, only the 2

- we can change an element of the list because it is mutable
- del(A[1])
- "hard rock".split() → string to list
- aliasing: multiple names referencing the same object (A = B)
- clone a list: B = A[:]
- help(A) for commands help
- B.sort() changes list B
  A = sorted(B) new list

Dictionaries

- type of collection
- key + value
  - key = index by label
- {"key":1, "key2":[3,3,3]}
- keys are immutable
- values can be
  - immutable
  - mutable
  - duplicates
- dict["key3"] = 44 → adds new value to de dict
- del(dict["key3"])
- verify if a key is in the dictionary: "key2" in dict
- dict.keys() → returns a list of keys
- dict.values() → returns a list of values

Sets

- type of collection
- input different input type
- unordered

- only unique elements
- set1 = {"hola", "rock", "rock"} → when the set is created, the duplicate values go away and the result is set1 = {"hola", "rock"}
- list to set → set(list1)
- set1.add("elemento")
- set1.remove("elemento")
- verify is an element is in the set: "elemento" in set1
- intesection: set1 & set2
- union: set1.union(set2)
- subset: set1.issubset(set2)

Semana 3

Conditions and branching

- Strings can be compared with == and ! =
- else if = elif
- not()
- or
- &

Loops

- range(3) → [0,1,2]
  range(10, 15) → [10, 11, 12, 13, 14]
- for i in range(0,5):
- for l in list1:
- for i, l in enumerate(list1):
- while(list1[i]=='orange'):

Functions

```
def function(a):
  #code
  return output

def NoWork():
  pass
#the function returns a None

#Collecting arguments
def ArtistNames(*names):
  for name in names:
```

Exception handling

```
try:
  getfile = open("myfile", "r")
  #
except IOError:
  #
except:
  #some other error -> not recommended
else:
  print("...successfull")
```

Objects and classes

Methods: functions that every instance of that class or type provides

```
class Circle(object):

  def __init__(self, radious, color): #constructor
    self.radious = radious
    self.color = color

  def add_radious(self, r):
    self.radious += r



#create instance
RedCircle = Circle(4, "red")
rad = RedCircle.radious

dir(nameOfObject) #returns list of data attributes and methods from a class
```

Semana 4

Reading files with open

```
file1 = open("path or name", "r") #w = write mode
file1.name
file1.mode
file1.close
```

It is better practice to use the WITH to open files because it closes them automatically

```
with open("Example1.txt", "r") as File1:
  file_stuff = File1.read()
  print(file_stuf)
  more_file_stuff = File1.readlines()
  more_file_stuff[0]
  more_file_stuff = File1.readline() #reads only first line
  more_file_stuff = File1.readline(5)
```

Writing files with open

```
Lines = ["una", "dos\n"]

with open("/path/to/file.txt", "w") as File1:
  File1.write("This is line A\n")
  for line in Lines:
    File1.write(line)
```

mode = "a" → append. Writes existing file

Nested withs

```
with open("asda", "r") as readfile:
  with open("as1", "r") as writefile:
    for line in readfile
      writefile.write(line)
```

Adittional modes

- **r+** : Reading and writing. Cannot truncate the file.
- **w+** : Writing and reading. Truncates the file.
- **a+** : Appending and Reading. Creates a new file, if none exists.

```
with open('Example2.txt', 'a+') as testwritefile:
    testwritefile.write("This is line E\n")
    print(testwritefile.read())
```

- `.tell()` - returns the current position in bytes
- `.seek(offset,from)` - changes the position by 'offset' bytes with respect to 'from'. From can take the value of 0,1,2
  corresponding to beginning, relative to current position and end

```
with open('Example2.txt', 'a+') as testwritefile:
    print("Initial Location: {}".format(testwritefile.tell()))

    data = testwritefile.read()
    if (not data):  #empty strings return false in python
            print('Read nothing')
    else:
            print(testwritefile.read())

    testwritefile.seek(0,0) # move 0 bytes from beginning.

    print("\nNew Location : {}".format(testwritefile.tell()))
    data = testwritefile.read()
    if (not data):
            print('Read nothing')
    else:
            print(data)

    print("Location after read: {}".format(testwritefile.tell()) )
```

```
with open('Example2.txt', 'r+') as testwritefile:
    data = testwritefile.readlines()
    testwritefile.seek(0,0) #write at beginning of file

    testwritefile.write("Line 1" + "\n")
    testwritefile.write("Line 2" + "\n")
    testwritefile.write("Line 3" + "\n")
    testwritefile.write("finished\n")
    #Uncomment the line below
    #testwritefile.truncate()
    testwritefile.seek(0,0)
    print(testwritefile.read())
```

Loading data with Pandas

```
import pandas as pd

csv_path = 'file1.csv'
df = pd.read_csv(csv_path)
#df = pd.read_excel(path)

df.head() #first 5 rows

#dataframe from dictonary
#keys = table headers, values are lists corresponding to rows
songs_frame = pd.DataFrame(songs_dict')
x = df[['Length']] #dataframe consisting of one column of the dict
```

Pandas: working with and saving data

```
df['ColumName'].unique() #returns all unique values

df['Released'] >= 1980 #returns a series of booleans
df1 = df[df['Released'] >= 1980]

df1.to_csv('new_songs.csv')
```

iloc, loc

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
0	Michael Jackson	Thriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
1	AC/DC	Back in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	9.5
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	9.0
3	Whitney Houston	The Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	8.0
5	Eagles	Their Greatest Hits (1971-1975)	1976	0:43:08	rock, soft rock, folk rock	32.2	42	17-Feb-76	NaN	7.5
6	Bee Gees	Saturday Night Fever	1977	1:15:54	disco	20.6	40	15-Nov-77	Y	7.0
7	Fleetwood Mac	Rumours	1977	0:40:01	soft rock	27.9	40	04-Feb-77	NaN	6.5

How do we select Albums The Dark Side of the Moon to Their Greatest Hits (1971-1975)? Select all that apply.

- [ ] df.iloc[2:5, 'Album']
- [x] df.loc[2:5, 'Album']

✓ **Correcto**
correct

- [x] df.iloc[2:6, 1]

✓ **Correcto**
correct

- [ ] df.loc[2:5, 1]

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
0	Michael Jackson	Thriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
1	AC/DC	Back in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	9.5
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	9.0
3	Whitney Houston	The Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	8.0
5	Eagles	Their Greatest Hits (1971-1975)	1976	0:43:08	rock, soft rock, folk rock	32.2	42	17-Feb-76	NaN	7.5
6	Bee Gees	Saturday Night Fever	1977	1:15:54	disco	20.6	40	15-Nov-77	Y	7.0
7	Fleetwood Mac	Rumours	1977	0:40:01	soft rock	27.9	40	04-Feb-77	NaN	6.5

How would you select the Genre disco? Select all that apply.

- [ ] df.iloc[6, 'genre']
- [ ] df.loc[6, 5]
- [x] df.iloc[6, 4]

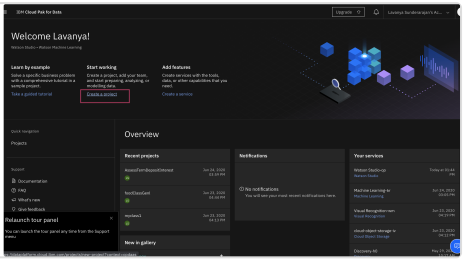
✓ **Correcto**
correct

- [ ] df.loc['Bee Gees', 'Genre']

In this lab you will explore Watson Studio

Objectives - By the end of this exercise, you will be able to: Create a Watson Studio service instance Create a project in Watson Studio Load a notebook in Watson Studio.

https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%204/PY0101EN-4-Lab-Loading%20Data%20and%20Viewing%20Data.md.html?origin=www.coursera.org



Coursera assignment - IBM Cloud Pak for Data

https://eu-de.dataplatform.cloud.ibm.com/analytics/notebooks/v2/4611a7e6-c37c-4485-bd17-9d01e9e6a0a4/view?access_token=524773625662626b4f5c94be137c2c40eda0a8b0f6daadb89dadf4ba8d4bf5dd

One dimensional Numpy

Numpy array

```
import numpy as np

#nd array
a = ["0", 1, "dos", 3]

type(a) #numpy.ndarray
a.dtype #dtype('int64')

a = np.array([0,1,2,3])
a.size
a.ndim #number of array dimensions
a.shape #size of the array in each dimension
```

Indexing and slicing methods

```
c = np.array([20, 1, 2, 3])

d = c[1:4] #slices from index 1 to 3

c[3:5] = 300, 400
```

Basic operations

```
#vector addition and sustraction
u = np.array([0,1])
v = np.array([2,3])
z = u+v = [2, 4]

#multiplication with a scalar
z = 2*u = [4, 6]

#entry-wise product
z = u*v = [0, 3]

#dot product a1*b1+a2*b2
result = np.dot(u,v)

#adding a constant (broadcasting)
z = u+1 #adds 1 to every entry

u.mean()
u.max()

np.pi
x = np.array([0, np.pi/2, np.pi])
y = np.sin(x)
```

```
np.linspace(-2, 2, num=5) #start of sequence, end of seq, number of samples
x = np.linspace(0, 2*np.pi, 100)

import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(x,y)
```

Plotting functions

```
# Import the libraries

import time
import sys
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

# Plotting functions

def Plotvec1(u, z, v):

    ax = plt.axes()
    ax.arrow(0, 0, *u, head_width=0.05, color='r', head_length=0.1)
    plt.text(*(u + 0.1), 'u')

    ax.arrow(0, 0, *v, head_width=0.05, color='b', head_length=0.1)
    plt.text(*(v + 0.1), 'v')
    ax.arrow(0, 0, *z, head_width=0.05, head_length=0.1)
    plt.text(*(z + 0.1), 'z')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)

def Plotvec2(a,b):
    ax = plt.axes()
    ax.arrow(0, 0, *a, head_width=0.05, color ='r', head_length=0.1)
    plt.text(*(a + 0.1), 'a')
    ax.arrow(0, 0, *b, head_width=0.05, color ='b', head_length=0.1)
    plt.text(*(b + 0.1), 'b')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)
```

Two dimensional Numpy

$$a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$$

$$A = np.array(a)$$

$$A: \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$$

```
A.ndim #dimensions
A.shape #(3 nested lists,3 size of each nested list or columns)
A.size #9

A[0,0] = A[0][0]

A[0:2, 2] #slicing

#multiplying matrices -> multiply elements in the same position
Z = X*Y

C = np.dot(A,B) #matrix multiplication -> row*column

C.T #transposed of C
```

```
1
2   X=np.array([[1,0],[0,1]])
3   Y=np.array([[2,1],[1,2]])
4   Z=np.dot(X,Y)
5
```

array([[2,1],[1,2]])

Semana 5

Simple APIs

What is it

- Your programm talks to other software components thorugh APIs
- Pandas is an API

REST APIs

- REpresentational State Transfer APIs
- interact with **web services**
- rules
  - communication
  - input or request
  - output or response
- **HTTP** methods are a way of transmitting data over the internet
  - send the rest api a request, usually via an http message
  - the http message is usually JSON

CryptoCurrency data

- **pycoingecko** Python Client/Wrapper for the Coin Gecko API
- updated every minute
- easy to use

Example: getting data on bitcoin in US Dollars for the past 30 days

```
!pip install pycoingecko
from pycoingecko import CoinGeckoAPI

cg = CoinGeckoAPI()
bitcoin_data = cg.get_coin_market_chart_by_id(id='bitcoin', vs_currency='usd', days=3
0)

#this returns a json. We are interested only in the prices
bitcoin_price_data = bitcoin_data['prices'] #this is a nested list

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

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

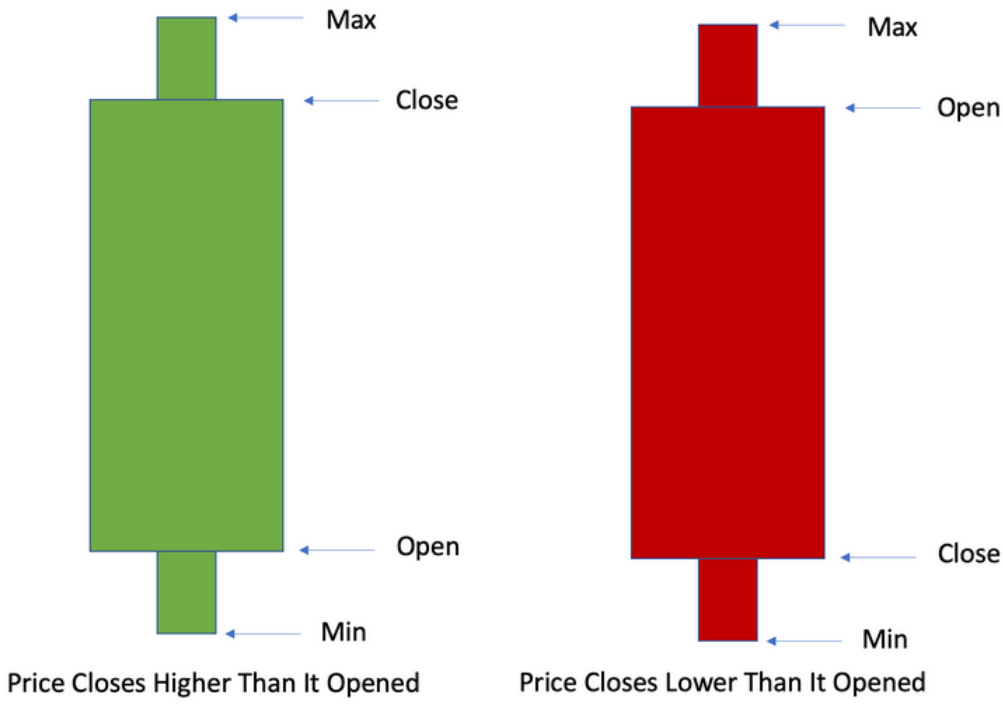
```
candlestick_data = data.groupby(data.Date.dt.date).agg({'Price': ['min', 'max', 'firs
t', 'last']})

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='Pr
ice (USD $)', title='Bitcon Candlestick Chart Over Past 30 Days')

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

Here is a description of the candle sticks.



API Key

way to access the API, unique set of characters to authorize YOU. You might get charged for every request, so this key shouldn't be shared

Endpoint

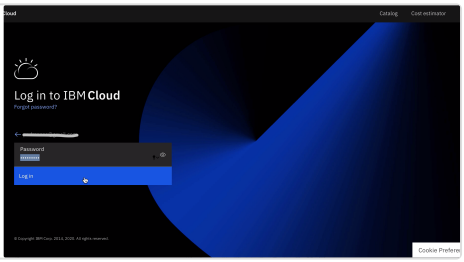
location of the API on the internet, just like a web address

Watson Speech to Text and Language translator

Step 3: Create a Speech to Text Instance

In this lab, you will learn: In this lab, you will create an instance of IBM Speech to Text and IBM Language Translator. IBM Speech to Text is a service that converts speech in audio format

https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/PY0101_Module5_Instructions_for_Speech_to_Text_and_Language_Translator_API_Keys.md.html?origin=www.coursera.org




```
pip install ibm_watson
```

```
from ibm_watson import SpeechToTextV1

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

```
filename = 'hello_this_is_python.wav'
with open(filename, mode="rb") as wav:
  response = s2t.recognize(audio=wav, content_type='audio/wav')
  #response.result returns a dictionary: {'results': [{'alternatives': [{'confidence':
0.91, 'transcript' 'hello this is python '}], 'final': True}], 'result_index':0}
  recognized_text = response.result['results'][0]["alternatives"][0]["transcript"]
  #now recognized_text = 'hello this is python '
```

```
from ibm_watson import LanguageTranslatorV3

url_lt = 'https://gateway.watsonplatform.net/language-translator/api'
apikey_lt = 'JKJBAkjb'
version_lt = '2018-05-01'

language_translator = LanguageTranslatorV3(iam_apikey=apikey_lt, url=url_lt, version=v
ersion_lt)

language_translator.list_identifiable_languages().get_result()

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

translation = translation_response.get_result()
spanish_translation = translation['translations'][0]['translation']
translation_new = language_translator.translate(text=spanish_translation, model_id='es
-en').get_result() #back to english
translation_eng = translation_new['translations'][0]['translation']
```

REST APIs and HTTP requests

HTTP protocol

- Client uses web browser → http request → web server → http response → client
- URL
  - scheme: protocol, for this lab HTTP://
  - internet address or base url: find the location (www.ibm.com)
  - route: location on the web server (/images/logos/logo.png)
- Request message

```
Request start line:
Get/index.html HTTP/1.0

Request header:
User-Agent: python-requests/2.21.0
```

```
Accept-Encoding:
gzip, deflate
```

- Response message

```
Response start line
HTTP/1.0 200 OK

Response Header
Server: Apache-Cache: UNCACHEABLE

Response Body
<!DOCTYPE html>
<html>
  <body>
    <h1> My First Heading </h1>
    <p> My first paragraph </p>
  </body>
</html>
```

200 is a status code that means OK, there are other status codes

  - HTTP methods
    - get
      - retrieves data from the server
    - post
      - submits data to server
    - put
      - updates data already on server
    - delete
      - deletes data from server

Requests library for HTTP in Python

```
import requests

url = 'https://wwww.ibm.com'
r = requests.get(url)
r.status_code # ex: 200
r.request.headers
```

```
r.request.body #if there is no body -> None
header = r.headers #dictionary

header['date']
header['Content-Type'] #'text/html; charset=UTF-8'
r.encoding #'UTF-8'
r.text[0:100] #display the html
```

```
#http://httpbin.org/get? Name=Joseph&ID=123
url_get = 'http://httpbin.org/get'
payload = {"name":"Joseph", "ID":"123"}
r = requests.get(url_get, params=payload)

r.url
r.request.body #in this case the body is in the URL so this returns None
r.status_code
r.text
r.headers['Content-Type']

r.json() #returns python dict
r.json()['args']
```

```
url_post = 'http://httpbin.org/post'
r_post = requests.post(url_post, data=payload)

r_post.url #http://httpbin.org/post
r.url #http://httpbin.org/get?name=Joseph&ID=123
```

Webscraping

Process that can be used to automatically extract information from a website.

We only need Python and the modules: Requests and BeautifulSoup

```
from bs4 import BeautifulSoup

html = "<!DOCTYPE html><html><head><title>Page Title</title></head><body><h3><b id
='boldest'>Lebron James</b></h3><p>Salaary: $92,000,000 </p></body></html>

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

With beautiful soup we can create:

Tag Object

tag_object=soup.title

tag_object:
<title>Page Title</title>

tag_object=soup.h3

tag_object:
<h3>Lebron James</h3>

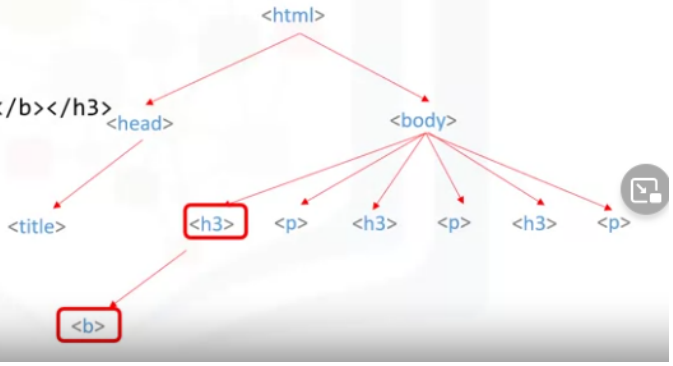
```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

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>
```

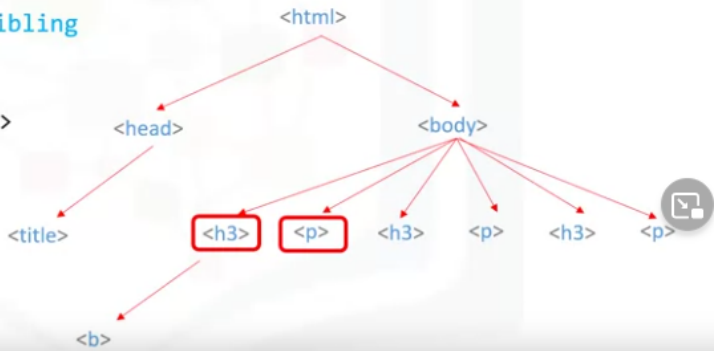


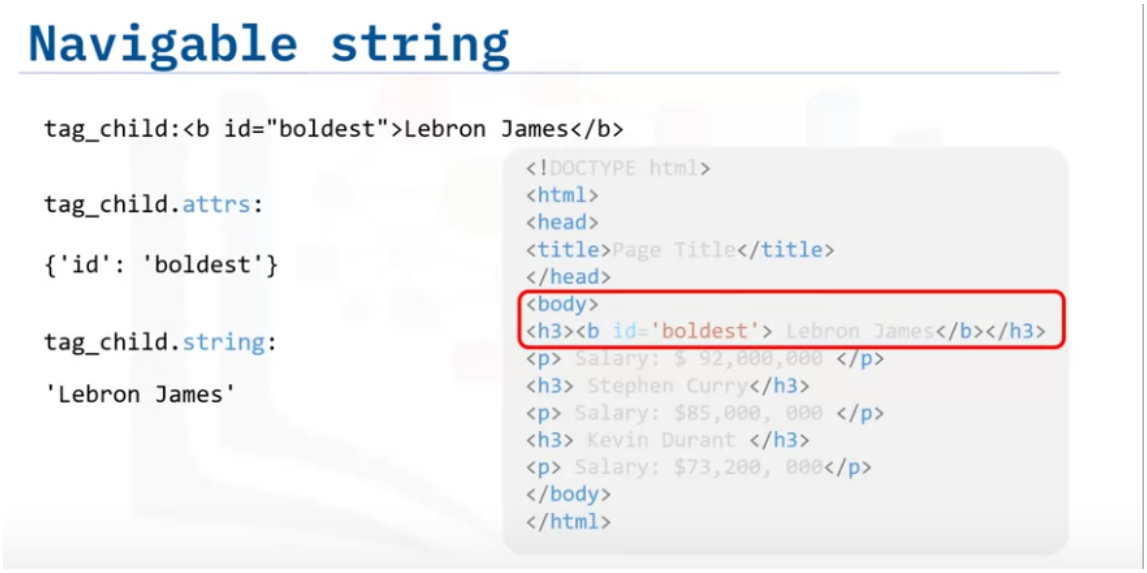
Next-sibling attribute

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

sibling_1= tag_object.next_sibling
sibling_1:

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






```
table.find_all(name='tr') #returns a list with every tr tag

#iterate through tables
for i, row  in enumerate(table_rows):
  cells = row.find_all("td")
  for j, cell in enumerate(cells):
    ###
```

Example

```
import requests
from bs4 import BeautifulSoup

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

soup = BeautifulSoup(page, "html.parser")
artists = soup.find_all('a')

for artist in artists:
  names = artist.contents[0]
  fullLink = artist.get('href')
```

Working with different file formats

```
import pandas as pd
import json
import xml.etree.ElementTree as etree

file = "FileExample.csv"
df = pd.read_csv(file)
```

```
#dont use the first line as header
df.columns = ['Name', 'Phone Number', 'Birthday'] #this is the header now

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

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