Python Cheat Sheet

Pandas | Numpy | Sklearn
Matplotlib | Seaborn
BS4 | Selenium | Scrapy

by Frank Andrade

Python Basics Cheat Sheet

Here you will find all the Python core concepts you need to know before learning any third-party library.

Data Types

```
Integers (int): 1
Float (float): 1.2
String (str): "Hello World"
Boolean: True/False
List: [value1, value2]
Dictionary: {key1:value1, key2:value2, ...}
```

Numeric Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponent
%	Modulus
//	Floor division

Comparison Operators

==	Equal to
!=	Different
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

String methods

```
string.upper(): converts to uppercase
string.lower(): converts to lowercase
string.title(): converts to title case
string.count('l'): counts how many times "l"
                   appears
string.find('h'): position of the "h" first
                  ocurrance
string.replace('o', 'u'): replaces "o" with "u"
```

Variables

Variable assignment:

```
message_1 = "I'm learning Python"
message_2 = "and it's fun!"
```

String concatenation (+ operator):

```
message_1 + ' ' + message_2
```

String concatenation (f-string):

```
f'{message_1} {message_2}'
```

List

Creating a list:

```
countries = ['United States', 'India',
             'China', 'Brazil']
```

Create an empty list:

```
my_list = []
```

Indexing:

```
>>> countries[0]
United States

>>> countries[3]
Brazil

>>> countries[-1]
Brazil
```

Slicing:

```
>>>countries[0:3]
['United States', 'India', 'China']

>>>countries[1:]
['India', 'China', 'Brazil']

>>>countries[:2]
['United States', 'India']
```

Adding elements to a list:

```
countries.append('Canada')
countries.insert(0,'Canada')
```

Nested list:

```
nested_list = [countries, countries_2]
```

Remove element:

```
countries.remove('United States')
countries.pop(0)#removes and returns value
del countries[0]
```

Creating a new list:

```
numbers = [4, 3, 10, 7, 1, 2]
```

Sorting a list:

```
>>> numbers.sort()
[1, 2, 3, 4, 7, 10]

>>> numbers.sort(reverse=True)
[10, 7, 4, 3, 2, 1]
```

Update value on a list:

```
>>> numbers[0] = 1000
>>> numbers
[1000, 7, 4, 3, 2, 1]
```

Copying a list:

```
new_list = countries[:]
new_list_2 = countries.copy()
```

Built-in Functions

Print an object:

```
print("Hello World")
```

Return the length of x:

```
len(x)
```

Return the minimum value:

```
min(x)
```

Return the maximum value:

```
max(x)
```

Returns a sequence of numbers:

```
range(x1,x2,n) # from x1 to x2
(increments by n)
```

Convert x to a string:

```
str(x)
```

Convert x to an integer/float:

```
int(x)
float(x)
```

Convert x to a list:

```
list(x)
```

Dictionary

Creating a dictionary:

```
my_data = {'name':'Frank', 'age':26}
```

Create an empty dictionary:

```
my_dict = {}
```

Get value of key "name":

```
>>> my_data["name"]
'Frank'
```

Get the keys:

```
>>> my_data.keys()
dict_keys(['name', 'age'])
```

Get the values:

```
>>> my_data.values()
dict_values(['Frank', 26])
```

Get the pair key-value:

```
>>> my_data.items()
dict_items([('name', 'Frank'), ('age', 26)])
```

Adding/updating items in a dictionary:

```
my_data['height']=1.7
my_data.update({'height':1.8,
          'languages':['English', 'Spanish']})
>>> my_data
{'name': 'Frank',
'age': 26,
'height': 1.8,
'languages': ['English', 'Spanish']}
```

Remove an item:

```
my_data.pop('height')
del my_data['languages']
my_data.clear()
```

Copying a dictionary:

```
new_dict = my_data.copy()
```

If Statement

Conditional test:

```
if <condition>:
    <code>
elif <condition>:
    <code>
...
else:
    <code>
```

Example:

```
if age>=18:
    print("You're an adult!")
```

Conditional test with list:

```
if <value> in <list>:
    <code>
```

Loops

For loop:

```
for <variable> in <list>:
    <code>
```

For loop and enumerate list elements:

```
for i, element in enumerate(<list>):
    <code>
```

For loop and obtain dictionary elements:

```
for key, value in my_dict.items():
    <code>
```

While loop:

```
while <condition>:
    <code>
```

Data Validation

Try-except:

```
try:
    <code>
except <error>:
    <code>
```

Loop control statement:

```
break: stops loop execution
continue: jumps to next iteration
pass: does nothing
```

Functions

Create a function:

```
def function(<params>):
    <code>
    return <data>
```

Modules

Import module:

```
import module
module.method()
```

OS module:

```
import os
os.getcwd()
os.listdir()
os.makedirs(<path>)
```

Special Characters

#	Comment
\n	New Line

Boolean Operators

and	logical AND
or	logical OR
not	logical NOT

Boolean Operators (Pandas)

&	logical AND
|	logical OR
~	logical NOT

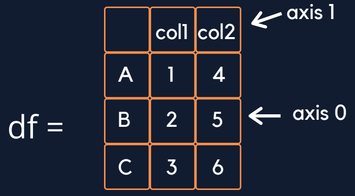
Below there are my guides, tutorials and complete Data Science course:

- Medium Guides
- YouTube Tutorials
- Data Science Course (Udemy)

Made by Frank Andrade frank-andrade.medium.com

Pandas Cheat Sheet

Pandas provides data analysis tools for Python. All of the following code examples refer to the dataframe below.



	col1	col2
A	1	4
B	2	5
C	3	6

Getting Started

Import pandas:

```
import pandas as pd
```

Create a series:

```
s = pd.Series([1, 2, 3],
              index=['A', 'B', 'C'],
              name='col1')
```

Create a dataframe:

```
data = [[1, 4], [2, 5], [3, 6]]
index = ['A', 'B', 'C']
df = pd.DataFrame(data, index=index,
                  columns=['col1', 'col2'])
```

Read a csv file with pandas:

```
df = pd.read_csv('filename.csv')
```

Advanced parameters:

```
df = pd.read_csv('filename.csv', sep=',',
                 names=['col1', 'col2'],
                 index_col=0,
                 encoding='utf-8',
                 nrows=3)
```

Selecting rows and columns

Select single column:

```
df['col1']
```

Select multiple columns:

```
df[['col1', 'col2']]
```

Show first n rows:

```
df.head(2)
```

Show last n rows:

```
df.tail(2)
```

Select rows by index values:

```
df.loc['A'] df.loc[['A', 'B']]
```

Select rows by position:

```
df.loc[1] df.loc[1:]
```

Data wrangling

Filter by value:

```
df[df['col1'] > 1]
```

Sort by one column:

```
df.sort_values('col1')
```

Sort by columns:

```
df.sort_values(['col1', 'col2'],
               ascending=[False, True])
```

Identify duplicate rows:

```
df.duplicated()
```

Identify unique rows:

```
df['col1'].unique()
```

Swap rows and columns:

```
df = df.transpose()
df = df.T
```

Drop a column:

```
df = df.drop('col1', axis=1)
```

Clone a data frame:

```
clone = df.copy()
```

Connect multiple data frames vertically:

```
df2 = df + 5 #new dataframe
pd.concat([df,df2])
```

Merge multiple data frames horizontally:

```
df3 = pd.DataFrame([[1, 7],[8,9]],
                   index=['B', 'D'],
           columns=['col1', 'col3'])
#df3: new dataframe
```

Only merge complete rows (INNER JOIN):

```
df.merge(df3)
```

Left column stays complete (LEFT OUTER JOIN):

```
df.merge(df3, how='left')
```

Right column stays complete (RIGHT OUTER JOIN):

```
df.merge(df3, how='right')
```

Preserve all values (OUTER JOIN):

```
df.merge(df3, how='outer')
```

Merge rows by index:

```
df.merge(df3,left_index=True,
         right_index=True)
```

Fill NaN values:

```
df.fillna(0)
```

Apply your own function:

```
def func(x):
    return 2**x
df.apply(func)
```

Arithmetics and statistics

Add to all values:

```
df + 10
```

Sum over columns:

```
df.sum()
```

Cumulative sum over columns:

```
df.cumsum()
```

Mean over columns:

```
df.mean()
```

Standard deviation over columns:

```
df.std()
```

Count unique values:

```
df['col1'].value_counts()
```

Summarize descriptive statistics:

```
df.describe()
```

Hierarchical indexing

Create hierarchical index:

```
df.stack()
```

Dissolve hierarchical index:

```
df.unstack()
```

Aggregation

Create group object:

```
g = df.groupby('col1')
```

Iterate over groups:

```
for i, group in g:
      print(i, group)
```

Aggregate groups:

```
g.sum()
g.prod()
g.mean()
g.std()
g.describe()
```

Select columns from groups:

```
g['col2'].sum()
g[['col2', 'col3']].sum()
```

Transform values:

```
 import math
 g.transform(math.log)
```

Apply a list function on each group:

```
def strsum(group):
 return ''.join([str(x) for x in group.value])

 g['col2'].apply(strsum)
```

```
Below there are my guides, tutorials
and complete Pandas course:
- Medium Guides
- YouTube Tutorials
- Pandas Course (Udemy)
```

Made by Frank Andrade frank-andrade.medium.com

Data export

Data as NumPy array:

```
df.values
```

Save data as CSV file:

```
df.to_csv('output.csv', sep=",")
```

Format a dataframe as tabular string:

```
df.to_string()
```

Convert a dataframe to a dictionary:

```
df.to_dict()
```

Save a dataframe as an Excel table:

```
df.to_excel('output.xlsx')
```

Pivot and Pivot Table

Read csv file 1:

```
df_gdp = pd.read_csv('gdp.csv')
```

The pivot() method:

```
df_gdp.pivot(index="year",
             columns="country",
             values="gdppc")
```

Read csv file 2:

```
df_sales=pd.read_excel(
         'supermarket_sales.xlsx')
```

Make pivot table:

```
df_sales.pivot_table(index='Gender',
                     aggfunc='sum')
```

Make a pivot tables that says how much male and female spend in each category:

```
df_sales.pivot_table(index='Gender',
               columns='Product line',
               values='Total',
               aggfunc='sum')
```

Visualization

The plots below are made with a dataframe with the shape of df_gdp (pivot() method)

Import matplotlib:

```
import matplotlib.pyplot as plt
```

Start a new diagram:

```
plt.figure()
```

Scatter plot:

```
df.plot(kind='scatter')
```

Bar plot:

```
df.plot(kind='bar',
        xlabel='data1',
        ylabel='data2')
```

Lineplot:

```
df.plot(kind='line',
       figsize=(8,4))
```

Boxplot:

```
df['col1'].plot(kind='box')
```

Histogram over one column:

```
df['col1'].plot(kind='hist',
                bins=3)
```

Piechart:

```
df.plot(kind='pie',
        y='col1',
        title='Population')
```

Set tick marks:

```
labels = ['A', 'B', 'C', 'D']
positions = [1, 2, 3, 4]
plt.xticks(positions, labels)
plt.yticks(positions, labels)
```

Label diagram and axes:

```
plt.title('Correlation')
plt.xlabel('Nunstück')
plt.ylabel('Slotermeyer')
```

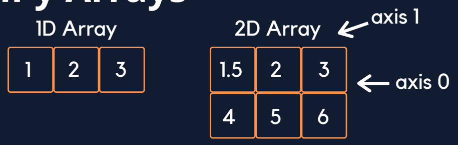
Save most recent diagram:

```
plt.savefig('plot.png')
plt.savefig('plot.png',dpi=300)
plt.savefig('plot.svg')
```

NumPy Cheat Sheet

NumPy provides tools for working with arrays. All of the following code examples refer to the arrays below.

NumPy Arrays



Getting Started

Import numpy:

```
  import numpy as np
```

Create arrays:

```
 a = np.array([1,2,3])
 b = np.array([(1.5,2,3), (4,5,6)], dtype=float)
 c = np.array([[(1.5,2,3), (4,5,6)],
              [(3,2,1), (4,5,6)]],
              dtype = float)
```

Initial placeholders:

```
 np.zeros((3,4)) #Create an array of zeros
 np.ones((2,3,4),dtype=np.int16)
 d = np.arange(10,25,5)
 np.linspace( 0,2, 9)
 e = np.full((2,2), 7)
 f = np.eye(2)
 np.random.random((2,2))
 np.empty((3,2))
```

Saving & Loading On Disk:

```
 np.save('my_array', a)
 np.savez('array.npz', a, b)
 np.load('my_array.npy')
```

Saving & Loading Text Files

```
 np.loadtxt('my_file.txt')
 np.genfromtxt('my_file.csv',
               delimiter=',')
 np.savetxt('myarray.txt', a,
            delimiter= ' ')
```

Inspecting Your Array

```
 a.shape
 len(a)
 b.ndim
 e.size
 b.dtype #data type
 b.dtype.name
 b.astype(int) #change data type
```

Data Types

```
 np.int64
 np.float32
 np.complex
 np.bool
 np.object
 np.string_
 np.unicode_
```

Array Mathematics

Arithmetic Operations

```
>>> g = a-b
 array([[-0.5, 0. , 0. ],
        [-3. , 3. , 3. ]])
>>> np.subtract(a,b)

>>> b+a
 array([[2.5, 4. , 6. ],
       [ 5. , 7. , 9. ]])
>>> np.add(b,a)

>>> a/b
 array([[ 0.66666667, 1. , 1. ],
       [ 0.2 5 , 0.4 , 0 . 5 ]])
>>> np.divide(a,b)

>>> a*b
 array([[ 1 . 5, 4. , 9. ],
        [ 4. , 10. , 18. ]])
>>> np.multiply(a,b)

>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.log(a)
>>> e.dot(f)
```

Aggregate functions:

```
 a.sum()
 a.min()
 b.max(axis= 0)
 b.cumsum(axis= 1) #Cumulative sum
 a.mean()
 b.median()
 a.corrcoef() #Correlation coefficient
 np.std(b) #Standard deviation
```

Copying arrays:

```
 h = a.view() #Create a view
 np.copy(a)
 h = a.copy() #Create a deep copy
```

Sorting arrays:

```
 a.sort() #Sort an array
 c.sort(axis=0)
```

Array Manipulation

Transposing Array:

```
 i = np.transpose(b)
 i.T
```

Changing Array Shape:

```
 b.ravel()
 g.reshape(3,-2)
```

Adding/removing elements:

```
 h.resize((2,6))
 np.append(h,g)
 np.insert(a, 1, 5)
 np.delete(a,[1])
```

Combining arrays:

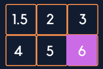
```
 np.concatenate((a,d),axis=0)
 np.vstack((a,b)) #stack vertically
 np.hstack((e,f)) #stack horizontally
```

Splitting arrays:

```
 np.hsplit(a,3) #Split horizontally
 np.vsplit(c,2) #Split vertically
```

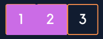
Subsetting

```
 b[1,2]
```



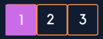
Slicing:

```
 a[0:2]
```



Boolean Indexing:

```
 a[a<2]
```



Scikit-Learn Cheat Sheet

Sklearn is a free machine learning library for Python. It features various classification, regression and clustering algorithms.

Getting Started

The code below demonstrates the basic steps of using sklearn to create and run a model on a set of data.

The steps in the code include loading the data, splitting into train and test sets, scaling the sets, creating the model, fitting the model on the data using the trained model to make predictions on the test set, and finally evaluating the performance of the model.

```
from sklearn import neighbors,datasets,preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = datasets.load_iris()
X,y = iris.data[:,:2], iris.target
X_train, X_test, y_train, y_test=train_test_split(X,y)
scaler = preprocessing_StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
knn = neighbors.KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy_score(y_test, y_pred)
```

Loading the Data

The data needs to be numeric and stored as NumPy arrays or SciPy spare matrix (numeric arrays, such as Pandas DataFrame's are also ok)

```
>>> import numpy as np
>>> X = np.random.random((10,5))
array([[0.21,0.33],
      [0.23, 0.60],
      [0.48, 0.62]])
>>> y = np.array(['A','B','A'])
array(['A', 'B', 'A'])
```

Training and Test Data

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,
random_state = 0)#Splits data into training and test set
```

Preprocessing The Data

Standardization

Standardizes the features by removing the mean and scaling to unit variance.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)
standarized_X = scaler.transform(X_train)
standarized_X_test = scaler.transform(X_test)
```

Normalization

Each sample (row of the data matrix) with at least one non-zero component is rescaled independently of other samples so that its norm equals one.

```
from sklearn.preprocessing import Normalizer
scaler = Normalizer().fit(X_train)
normalized_X = scaler.transform(X_train)
normalized_X_test = scaler.transform(X_test)
```

Binarization

Binarize data (set feature values to 0 or 1) according to a threshold.

```
from sklearn.preprocessing import Binarizer
binarizer = Binarizer(threshold = 0.0).fit(X)
binary_X = binarizer.transform(X_test)
```

Encoding Categorical Features

Imputation transformer for completing missing values.

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit_transform(X_train)
```

Imputing Missing Values

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=0, strategy ='mean')
imp.fit_transform(X_train)
```

Generating Polynomial Features

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(5)
poly.fit_transform(X)
```

Create Your Model

Supervised Learning Models

Linear Regression

```
from sklearn.linear_model import LinearRegression
lr  = LinearRegression(normalize = True)
```

Support Vector Machines (SVM)

```
from sklearn.svm import SVC
svc = SVC(kernel = 'linear')
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

KNN

```
from sklearn import neighbors
knn = neighbors.KNeighborsClassifier(n_neighbors = 5)
```

Unsupervised Learning Models

Principal Component Analysis (PCA)

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 0.95)
```

K means

```
from sklearn.cluster import KMeans
k_means = KMeans(n_clusters = 3, random_state = 0)
```

Model Fitting

Fitting supervised and unsupervised learning models onto data.

Supervised Learning

```
lr.fit(X, y) #Fit the model to the data
knn.fit(X_train,y_train)
svc.fit(X_train,y_train)
```

Unsupervised Learning

```
k_means.fit(X_train) #Fit the model to the data
pca_model = pca.fit_transform(X_train)#Fit to data,then transform
```

Prediction

Predict Labels

```
y_pred = lr.predict(X_test) #Supervised Estimators
y_pred = k_means.predict(X_test) #Unsupervised Estimators
```

Estimate probability of a label

```
y_pred = knn.predict_proba(X_test)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
knn.score(X_test,y_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

Confusion Matrix

```
from sklearn .metrics import confusion_matrix
print(confusion_matrix(y_test,y_pred))
```

Regression Metrics

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test,y_pred)
```

Mean Squared Error

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,y_pred)
```

R^2 Score

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
from sklearn.metrics import adjusted_rand_score
adjusted_rand_score(y_test,y_pred)
```

Homogeneity

```
from sklearn.metrics import homogeneity_score
homogeneity_score(y_test,y_pred)
```

V-measure

```
from sklearn.metrics import v_measure_score
v_measure_score(y_test,y_pred)
```

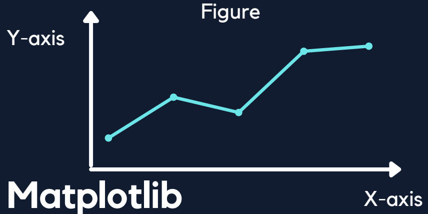
Tune Your Model

Grid Search

```
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':np.arange(1,3),
          'metric':['euclidean','cityblock']}
grid = GridSearchCV(estimator = knn, param_grid = params)
grid.fit(X_train, y_train)
print(grid.best_score_)
print(grid.best_estimator_)
```

Data Viz Cheat Sheet

Matplotlib is a Python 2D plotting library that produces figures in a variety of formats.



Matplotlib

Workflow

The basic steps to creating plots with matplotlib are Prepare Data, Plot, Customize Plot, Save Plot and Show Plot.

```
  import matplotlib.pyplot as plt
```

Example with lineplot

Prepare data

```
    x = [2017, 2018, 2019, 2020, 2021]
    y = [43, 45, 47, 48, 50]
```

Plot & Customize Plot

```
    plt.plot(x,y,marker='o',linestyle='--',
    color='g', label='USA')
    plt.xlabel('Years')
    plt.ylabel('Population (M)')
    plt.title('Years vs Population')
    plt.legend(loc='lower right')
    plt.yticks([41, 45, 48, 51])
```

Save Plot

```
    plt.savefig('example.png')
```

Show Plot

```
    plt.show()
```

Markers: '.', 'o', 'v', '<', '>'

Line Styles: '-', '--', '-.', ':'

Colors: 'b', 'g', 'r', 'y' #blue, green, red, yellow

Barplot

```
 x = ['USA', 'UK', 'Australia']
 y = [40, 50, 33]
 plt.bar(x, y)
 plt.show()
```

Piechart

```
 plt.pie(y, labels=x, autopct='%.0f %%')
 plt.show()
```

Histogram

```
 ages = [15, 16, 17, 30, 31, 32, 35]
 bins = [15, 20, 25, 30, 35]
 plt.hist(ages, bins, edgecolor='black')
 plt.show()
```

Boxplots

```
 ages = [15, 16, 17, 30, 31, 32, 35]
 plt.boxplot(ages)
 plt.show()
```

Scatterplot

```
 a = [1, 2, 3, 4, 5, 4, 3 ,2, 5, 6, 7]
 b = [7, 2, 3, 5, 5, 7, 3, 2, 6, 3, 2]
 plt.scatter(a, b)
 plt.show()
```

Subplots

Add the code below to make multple plots with 'n' number of rows and columns.

```
fig, ax = plt.subplots(nrows=1,
                       ncols=2,
                       sharey=True,
                       figsize=(12, 4))
```

Plot & Customize Each Graph

```
 ax[0].plot(x, y, color='g')
 ax[0].legend()
 ax[1].plot(a, b, color='r')
 ax[1].legend()
 plt.show()
```

Seaborn

Workflow

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

Lineplot

```
 plt.figure(figsize=(10, 5))
 flights = sns.load_dataset("flights")
 may_flights=flights.query("month=='May'")
 ax = sns.lineplot(data=may_flights,
                   x="year",
                   y="passengers")
 ax.set(xlabel='x', ylabel='y',
        title='my_title, xticks=[1,2,3])
 ax.legend(title='my_legend,
           title_fontsize=13)
 plt.show()
```

Barplot

```
 tips = sns.load_dataset("tips")
 ax = sns.barplot(x="day",
                  y="total_bill,
                  data=tips)
```

Histogram

```
 penguins = sns.load_dataset("penguins")
 sns.histplot(data=penguins,
              x="flipper_length_mm")
```

Boxplot

```
 tips = sns.load_dataset("tips")
 ax = sns.boxplot(x=tips["total_bill"])
```

Scatterplot

```
  tips = sns.load_dataset("tips")
  sns.scatterplot(data=tips,
                  x="total_bill",
                  y="tip")
```

Figure aesthetics

```
 sns.set_style('darkgrid') #stlyes
 sns.set_palette('husl', 3) #palettes
 sns.color_palette('husl') #colors
```

Fontsize of the axes title, x and y labels, tick labels and legend:

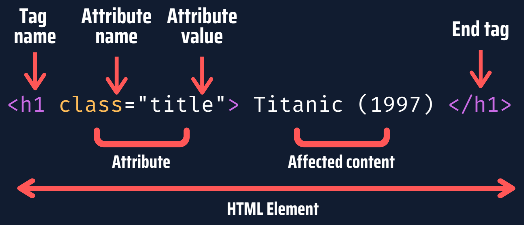
```
 plt.rc('axes', titlesize=18)
 plt.rc('axes', labelsize=14)
 plt.rc('xtick', labelsize=13)
 plt.rc('ytick', labelsize=13)
 plt.rc('legend', fontsize=13)
 plt.rc('font', size=13)
```

Web Scraping Cheat Sheet

Web Scraping is the process of extracting data from a website. Before studying Beautiful Soup and Selenium, it's good to review some HTML basics first.

HTML for Web Scraping

Let's take a look at the HTML element syntax.

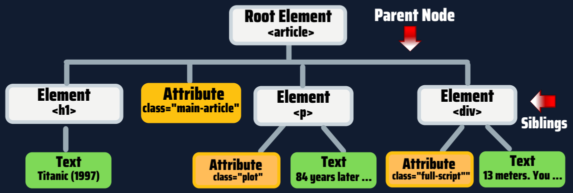


This is a single HTML element, but the HTML code behind a website has hundreds of them.

HTML code example

```
<article class="main-article">
 <h1> Titanic (1997) </h1>
 <p class="plot"> 84 years later ... </p>
 <div class="full-script"> 13 meters. You ... </div>
</article>
```

The HTML code is structured with "nodes". Each rectangle below represents a node (element, attribute and text nodes)



- "Siblings" are nodes with the same parent.
- A node's children and its children's children are called its "descendants". Similarly, a node's parent and its parent's parent are called its "ancestors".
- it's recommended to find element in this order.
  a. ID
  b. Class name
  c. Tag name
  d. Xpath

Beautiful Soup

Workflow

Importing the libraries

```
from bs4 import BeautifulSoup
import requests
```

Fetch the pages

```
result=requests.get("www.google.com")
result.status_code #get status code
result.headers #get the headers
```

Page content

```
content = result.text
```

Create soup

```
soup = BeautifulSoup(content,"lxml")
```

HTML in a readable format

```
print(soup.prettify())
```

Find an element

```
soup.find(id="specific_id")
```

Find elements

```
soup.find_all("a")
soup.find_all("a","css_class")
soup.find_all("a",class_="my_class")
soup.find_all("a",attrs={"class":
                         "my_class"})
```

Get inner text

```
sample = element.get_text()
sample = element.get_text(strip=True,
                          separator= ' ')
```

Get specific attributes

```
sample = element.get('href')
```

XPath

We need to learn XPath to scrape with Selenium or Scrapy.

XPath Syntax

An XPath usually contains a tag name, attribute name, and attribute value.

```
//tagName[@AttributeName="Value"]
```

Let's check some examples to locate the article, title, and transcript elements of the HTML code we used before.

```
//article[@class="main-article"]
//h1
//div[@class="full-script"]
```

XPath Functions and Operators

XPath functions

```
//tag[contains(@AttributeName, "Value")]
```

XPath Operators: and, or

```
//tag[(expression 1) and (expression 2)]
```

XPath Special Characters

/	Selects the children from the node set on the left side of this character
//	Specifies that the matching node set should be located at any level within the document
.	Specifies the current context should be used (refers to present node)
..	Refers to a parent node
*	A wildcard character that selects all elements or attributes regardless of names
@	Select an attribute
()	Grouping an XPath expression
[n]	Indicates that a node with index "n" should be selected

Selenium

Workflow

```
from selenium import webdriver
web="www.google.com"
path='introduce chromedriver path'
driver = webdriver.Chrome(path)
driver.get(web)
```

Find an element

```
driver.find_element_by_id('name')
```

Find elements

```
driver.find_elements_by_class_name()
driver.find_elements_by_css_selector
driver.find_elements_by_xpath()
driver.find_elements_by_tag_name()
driver.find_elements_by_name()
```

Quit driver

```
driver.quit()
```

Getting the text

```
data = element.text
```

Implicit Waits

```
import time
time.sleep(2)
```

Explicit Waits

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

WebDriverWait(driver, 5).until(EC.element_to_be_clickable((By.ID,
'id_name'))) #Wait 5 seconds until an element is clickable
```

Options: Headless mode, change window size

```
from selenium.webdriver.chrome.options import Options
options = Options()
options.headless = True
options.add_argument('window-size=1920x1080')
driver=webdriver.Chrome(path,options=options)
```

Below there are my guides, tutorials and complete web scraping course:
- Medium Guides
- YouTube Tutorials
- Web Scraping Course (Udemy)

Made by Frank Andrade frank-andrade.medium.com

Scrapy

Scrapy is the most powerful web scraping framework in Python, but it's a bit complicated to set up, so check my guide or its documentation to set it up.

Creating a Project and Spider

To create a new project, run the following command in the terminal.

```
scrapy startproject my_first_spider
```

To create a new spider, first change the directory.

```
cd my_first_spider
```

Create an spider

```
scrapy genspider example example.com
```

The Basic Template

When you create a spider, you obtain a template with the following content.

```
import scrapy
class ExampleSpider(scrapy.Spider):
    name = 'example'
    allowed_domains = ['example.com']          Class
    start_urls = ['http://example.com/']

    def parse(self, response):               Parse method
        pass
```

The class is built with the data we introduced in the previous command, but the parse method needs to be built by us. To build it, use the functions below.

Finding elements

To find elements in Scrapy, use the response argument from the parse method

```
response.xpath('//tag[@AttributeName="Value"]')
```

Getting the text

To obtain the text element we use text() and either .get() or .getall(). For example:

```
response.xpath(‘//h1/text()’).get()
response.xpath(‘//tag[@Attribute=”Value”]/text()’).getall()
```

Return data extracted

To see the data extracted we have to use the yield keyword

```
def parse(self, response):
  title = response.xpath(‘//h1/text()’).get()

  # Return data extracted
  yield {'titles': title}
```

Run the spider and export data to CSV or JSON

```
scrapy crawl example
scrapy crawl example -o name_of_file.csv
scrapy crawl example -o name_of_file.json
```