## Exercise1

# Importing flat files from the web: your turn!

You are about to import your first file from the web! The flat file you will import will be 'winequality-red.csv' from the University of California, Irvine's Machine Learning repository. The flat file contains tabular data of physiochemical properties of red wine, such as pH, alcohol content and citric acid content, along with wine quality rating.

The URL of the file is

`'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv'`

After you import it, you'll check your working directory to confirm that it is there and then you'll load it into a pandas DataFrame.

## Instructions

- Import the function urlretrieve from the subpackage urllib.request.
- Assign the URL of the file to the variable url.
- Use the function urlretrieve() to save the file locally as 'winequality-red.csv'.
- Execute the remaining code to load 'winequality-red.csv' in a pandas DataFrame and to print its head to the shell.

In [ ]:
```
# Import package
from ____ import ____

# Import pandas
import pandas as pd

# Assign url of file: url


# Save file locally


# Read file into a DataFrame and print its head
df = pd.read_csv('winequality-red.csv', sep=';')
print(df.head())


#_____________________#
#Solutions

# Import package
from urllib.request import urlretrieve

# Import pandas
import pandas as pd

# Assign url of file: url
url = 'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv'

# Save file locally
urlretrieve(url,'winequality-red.csv')

# Read file into a DataFrame and print its head
df = pd.read_csv('winequality-red.csv', sep=';')
print(df.head())
```

## Exercise2

# Opening and reading flat files from the web

You have just imported a file from the web, saved it locally and loaded it into a DataFrame. If you just wanted to load a file from the web into a DataFrame without first saving it locally, you can do that easily using pandas. In particular, you can use the function pd.read_csv() with the URL as the first argument and the separator sep as the second argument.

The URL of the file, once again, is

`'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv'`

## Instructions

- Assign the URL of the file to the variable url.
- Read file into a DataFrame df using pd.read_csv(), recalling that the separator in the file is ';'.
- Print the head of the DataFrame df.
- Execute the rest of the code to plot histogram of the first feature in the DataFrame df.

In [ ]:
```
# Import packages
import matplotlib.pyplot as plt
import pandas as pd

# Assign url of file: url


# Read file into a DataFrame: df


# Print the head of the DataFrame
print(____)

# Plot first column of df
pd.DataFrame.hist(df.ix[:, 0:1])
plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')
plt.ylabel('count')
plt.show()



#_____________________#
#Solutions

# Import packages
import matplotlib.pyplot as plt
import pandas as pd

# Assign url of file: url
url = 'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv'

# Read file into a DataFrame: df
df = pd.read_csv(url,sep=';')

# Print the head of the DataFrame
print(df.head())

# Plot first column of df
pd.DataFrame.hist(df.ix[:, 0:1])
plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')
plt.ylabel('count')
plt.show()
```

## Exercise3

# Importing non-flat files from the web

Congrats! You've just loaded a flat file from the web into a DataFrame without first saving it locally using the pandas function pd.read_csv(). This function is super cool because it has close relatives that allow you to load all types of files, not only flat ones. In this interactive exercise, you'll use pd.read_excel() to import an Excel spreadsheet.

The URL of the spreadsheet is

```
'http://s3.amazonaws.com/assets.datacamp.com/course/importing_data_into_r/latitude.xls'
```

Your job is to use pd.read_excel() to read in all of its sheets, print the sheet names and then print the head of the first sheet using its name, not its index.

Note that the output of pd.read_excel() is a Python dictionary with sheet names as keys and corresponding DataFrames as corresponding values.

## Instructions

- Assign the URL of the file to the variable url.
- Read the file in url into a dictionary xls using pd.read_excel() recalling that, in order to import all sheets you need to pass None to the argument sheet_name.
- Print the names of the sheets in the Excel spreadsheet; these will be the keys of the dictionary xls.
- Print the head of the first sheet using the sheet name, not the index of the sheet! The sheet name is '1700'

In [ ]:
```
# Import package
import pandas as pd

# Assign url of file: url


# Read in all sheets of Excel file: xls


# Print the sheetnames to the shell


# Print the head of the first sheet (using its name, NOT its index)




#_____________________#
#Solutions

# Import package
import pandas as pd

# Assign url of file: url
url = 'http://s3.amazonaws.com/assets.datacamp.com/course/importing_data_into_r/latitude.xls'

# Read in all sheets of Excel file: xls
xls = pd.read_excel(url, sheet_name=None)

# Print the sheetnames to the shell
print(xls.keys())

# Print the head of the first sheet (using its name, NOT its index)
print(xls['1700'].head())
```

## Exercise4

# Performing HTTP requests in Python using urllib

Now that you know the basics behind HTTP GET requests, it's time to perform some of your own. In this interactive exercise, you will ping our very own DataCamp servers to perform a GET request to extract information from the first coding exercise of this course,

`"https://campus.datacamp.com/courses/1606/4135?ex=2"` .

In the next exercise, you'll extract the HTML itself. Right now, however, you are going to package and send the request and then catch the response.

## Instructions

- Import the functions urlopen and Request from the subpackage urllib.request.
- Package the request to the url `"https://campus.datacamp.com/courses/1606/4135?ex=2"` using the function Request() and assign it to request.
- Send the request and catch the response in the variable response with the function urlopen().
- Run the rest of the code to see the datatype of response and to close the connection!

In [ ]:
```
# Import packages


# Specify the url
url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request: request


# Sends the request and catches the response: response


# Print the datatype of response
print(type(response))

# Be polite and close the response!
response.close()



#_____________________#
#Solutions

# Import packages
from urllib.request import urlopen,Request

# Specify the url
url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request: request
request= Request(url)

# Sends the request and catches the response: response
response = urlopen(request)

# Print the datatype of response
print(type(response))

# Be polite and close the response!
response.close()
```

## Exercise5

# Printing HTTP request results in Python using urllib

You have just packaged and sent a GET request to

"https://campus.datacamp.com/courses/1606/4135?ex=2"

and then caught the response. You saw that such a response is a http.client.HTTPResponse object. The question remains: what can you do with this response?

Well, as it came from an HTML page, you could read it to extract the HTML and, in fact, such a http.client.HTTPResponse object has an associated read() method. In this exercise, you'll build on your previous great work to extract the response and print the HTML.

## Instructions

- Send the request and catch the response in the variable response with the function urlopen(), as in the previous exercise.
- Extract the response using the read() method and store the result in the variable html.
- Print the string html.
- Hit submit to perform all of the above and to close the response: be tidy!

In [ ]:
```
# Import packages
from urllib.request import urlopen, Request

# Specify the url
url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request
request = Request(url)

# Sends the request and catches the response: response


# Extract the response: html


# Print the html


# Be polite and close the response!
response.close()


#_____________________#
#Solutions

# Import packages
from urllib.request import urlopen, Request

# Specify the url
url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request
request = Request(url)

# Sends the request and catches the response: response
response = urlopen(request)

# Extract the response: html
html = response.read()

# Print the html
print(html)

# Be polite and close the response!
response.close()
```

## Exercise6

# Performing HTTP requests in Python using requests

Now that you've got your head and hands around making HTTP requests using the urllib package, you're going to figure out how to do the same using the higher-level requests library. You'll once again be pinging DataCamp servers for their

`"http://www.datacamp.com/teach/documentation"` page.

Note that unlike in the previous exercises using urllib, you don't have to close the connection when using requests!

## Instructions

- Import the package requests.
- Assign the URL of interest to the variable url.
- Package the request to the URL, send the request and catch the response with a single function requests.get(), assigning the response to the variable r.
- Use the text attribute of the object r to return the HTML of the webpage as a string; store the result in a variable text.
- Hit submit to print the HTML of the webpage.

In [ ]:
```
# Import package


# Specify the url: url


# Packages the request, send the request and catch the response: r


# Extract the response: text


# Print the html
print(text)


#_____________________#
#Solutions

# Import packages
import requests

# Specify the url: url
url = "http://www.datacamp.com/teach/documentation"

# Packages the request, send the request and catch the response: r
r = requests.get(url)

# Extract the response: text
text= r.text

# Print the html
print(text)
```

## Exercise7

# Parsing HTML with BeautifulSoup

In this interactive exercise, you'll learn how to use the BeautifulSoup package to parse, prettify and extract information from HTML. You'll scrape the data from the webpage of Guido van Rossum, Python's very own Benevolent Dictator for Life. In the following exercises, you'll prettify the HTML and then extract the text and the hyperlinks.

The URL of interest is url = `'https://www.python.org/~guido/'` .

## Instructions

- Import the function BeautifulSoup from the package bs4.
- Assign the URL of interest to the variable url.
- Package the request to the URL, send the request and catch the response with a single function requests.get(), assigning the response to the variable r.
- Use the text attribute of the object r to return the HTML of the webpage as a string; store the result in a variable html_doc.
- Create a BeautifulSoup object soup from the resulting HTML using the function BeautifulSoup().
- Use the method prettify() on soup and assign the result to pretty_soup.
- Hit submit to print to prettified HTML to your shell!

In [ ]:
```
# Import packages
import requests
from ____ import ____

# Specify url: url


# Package the request, send the request and catch the response: r


# Extracts the response as html: html_doc


# Create a BeautifulSoup object from the HTML: soup


# Prettify the BeautifulSoup object: pretty_soup


# Print the response
print(pretty_soup)


#_____________________#
#Solutions

# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url: url
url ='https://www.python.org/~guido/'

# Package the request, send the request and catch the response:
r =requests.get(url)


# Extracts the response as html: html_doc
html_doc = r.text

# Create a BeautifulSoup object from the HTML: soup
soup= BeautifulSoup(html_doc)

# Prettify the BeautifulSoup object: pretty_soup
pretty_soup= soup.prettify()

# Print the response
print(pretty_soup)
```

## Exercise8

# Turning a webpage into data using BeautifulSoup: getting the text

As promised, in the following exercises, you'll learn the basics of extracting information from HTML soup. In this exercise, you'll figure out how to extract the text from the BDFL's webpage, along with printing the webpage's title.

## Instructions

- In the sample code, the HTML response object html_doc has already been created: your first task is to Soupify it using the function BeautifulSoup() and to assign the resulting soup to the variable soup.
- Extract the title from the HTML soup soup using the attribute title and assign the result to guido_title.
- Print the title of Guido's webpage to the shell using the print() function.
- Extract the text from the HTML soup soup using the method get_text() and assign to guido_text.
- Hit submit to print the text from Guido's webpage to the shell.

In [ ]:
```
# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url: url
url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r
r = requests.get(url)

# Extract the response as html: html_doc
html_doc = r.text

# Create a BeautifulSoup object from the HTML: soup


# Get the title of Guido's webpage: guido_title


# Print the title of Guido's webpage to the shell


# Get Guido's text: guido_text


# Print Guido's text to the shell
print(guido_text)

#_____________________#
#Solutions

# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url: url
url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r
r = requests.get(url)

# Extract the response as html: html_doc
html_doc = r.text

# Create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)

# Get the title of Guido's webpage: guido_title
guido_title = soup.title

# Print the title of Guido's webpage to the shell
print(guido_title)

# Get Guido's text: guido_text
guido_text = soup.get_text()

# Print Guido's text to the shell
print(guido_text)
```

## Exercise9

# Turning a webpage into data using BeautifulSoup: getting the hyperlinks

In this exercise, you'll figure out how to extract the URLs of the hyperlinks from the BDFL's webpage. In the process, you'll become close friends with the soup method find_all().

## Instructions

- Use the method find_all() to find all hyperlinks in soup, remembering that hyperlinks are defined by the HTML tag `<a>` but passed to find_all() without angle brackets; store the result in the variable a_tags.
- The variable a_tags is a results set: your job now is to enumerate over it, using a for loop and to print the actual URLs of the hyperlinks; to do this, for every element link in a_tags, you want to print() link.get('href').

In [ ]:
```
# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url
url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r
r = requests.get(url)

# Extracts the response as html: html_doc
html_doc = r.text

# create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)

# Print the title of Guido's webpage
print(soup.title)

# Find all 'a' tags (which define hyperlinks): a_tags


# Print the URLs to the shell
for ____ in ____:
    ____


#_____________________#
#Solutions

# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url
url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r
r = requests.get(url)

# Extracts the response as html: html_doc
html_doc = r.text

# create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)

# Print the title of Guido's webpage
print(soup.title)

# Find all 'a' tags (which define hyperlinks): a_tags
a_tags = soup.find_all('a')

# Print the URLs to the shell
for link in a_tags:
    print(link.get('href'))
```