

# DATA SCIENCE FUNDAMENTALS

## LESSON 9

Hay Kranen  
Monday October 21st, 2018



# TODAY'S PROGRAMME

Recap  
Reddit Plots  
Break  
Functions  
Web scraping  
Lunch break

# RECAP

```
CLIENT_ID = "Y0idhGvtxNY0oA"
CLIENT_SECRET = "_8nZnFMxafai7a0Mihqy8PCrBIc"
USER_AGENT = f"python:[{CLIENT_ID}]:0.1 (by /u/Husky)"
```

```
CLIENT_ID = "x"
CLIENT_SECRET = "x"
USER_AGENT = f"python:{CLIENT_ID}:0.1 (by /u/x)"
```

```
In [37]: print(newlist)
```

```
[{'title': "What was the cringiest thing you did in your first relationship?", 'score': 25260, 'comments': 10213}, {'title': "What's the dumbest thing you've heard someone say that made you wonder how they function on a day to day basis?", 'score': 50720, 'comments': 28783}, {'title': "What is something that HAS aged well?", 'score': 5140, 'comments': 6161}, {'title': "What have we all conveniently forgotten about?", 'score': 5058, 'comments': 4823}, {'title': "What made you laugh the hardest in your entire life?", 'score': 7745, 'comments': 3777}, {'title': "What is the worst thing to cheap out on?", 'score': 477, 'comments': 713}, {"title": "What's an experience you don't ever want to go through again?", 'score': 4880, 'comments': 4783}, {'title': "Men of reddit, what do women do that they think is sexy, but actually isn't?", 'score': 546, 'comments': 1314}, {'title': "What benefits did you notice after losing weight?", 'score': 1549, 'comments': 1278}, {"title": "What is the most physically painful thing you've ever gone through?", 'score': 231, 'comments': 615}]
```

```
df = pd.DataFrame({'mass': [score, score_total]},  
                  index=['High', 'Total score'])  
plot = df.plot.pie(y='mass', figsize=(15, 15), fontsize=40)  
  
fig = plot.get_figure()  
fig.savefig('plot.pdf')
```

# REDDIT PLOTS

# FUNCTIONS

A way to group code  
Reuse common functionality  
Local scope for variables

# Simple functions

```
def say_hello():
    print("Hello!")

say_hello()
```

**def say\_hello()**

def keyword

function name

# Functions with arguments

```
def say_hello(name):  
    print(f"Hello, {name}!")  
  
say_hello("Barrie")
```

**def say\_hello(name)**

**def keyword**      **function name**      **argument(s)**

# Return statements

```
import json

def read_json(path):
    with open(path) as f:
        data = json.load(f)

    return data

paintings = read_json("paintings.json")
```

**def read\_json(path)**

def keyword

function name

argument(s)

# Multiple arguments

```
import json

def write_json(data, path):
    with open(path, "w") as f:
        json_data = json.dumps(data)
        f.write(json_data)

colors = ["red", "green", "blue"]
write_json(colors, "colors.json")
```

# Default argument values

```
import json

def write_json(data, path = "data.json"):
    with open(path, "w") as f:
        json_data = json.dumps(data)
        f.write(json_data)

colors = ["red", "green", "blue"]
write_json(colors)
```

# functions

`print()`

`len()`

`open()`

# methods

`json.loads()`

`requests.get()`

# pandas.read\_json

```
pandas.read_json(path_or_buf=None, orient=None, typ='frame', dtype=True, convert_axes=True,  
convert_dates=True, keep_default_dates=True, numpy=False, precise_float=False, date_unit=None,  
encoding=None, lines=False, chunksize=None, compression='infer')  
[source]
```

Convert a JSON string to pandas object

**path\_or\_buf** : a valid JSON string or file-like, default: None

The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. For instance, a local file could be

`file://localhost/path/to/table.json`

**orient** : string,

Indication of expected JSON string format. Compatible JSON strings can be produced by `to_json()` with a corresponding orient value. The set of possible orients is:

- 'split' : dict like {index -> [index], columns -> [columns], data -> [values]}
- 'records' : list like [{column -> value}, ... , {column -> value}]

```
import json

def write_json(data, path):
    with open(path, "w") as f:
        json_data = json.dumps(data)
        f.write(json_data)

colors = ["red", "green", "blue"]
write_json(colors, "colors.json")
```

# Keyword arguments

```
import json

def write_json(data, path):
    with open(path, "w") as f:
        json_data = json.dumps(data)
        f.write(json_data)

colors = ["red", "green", "blue"]
write_json(path = "colors.json", data = colors)
```

# Functions calling other functions

```
import json

def print_json(path):
    json_data = read_json(path)
    json_str = json.dumps(json_data, indent = 4)
    print(json_str)

def read_json(path):
    with open(path) as f:
        data = json.load(f)

    return data

print_json("footballers.json")
```

```
import json

def print_json(path):
    json_data = read_json(path)
    json_str = json.dumps(json_data, indent = 4)
    print(json_str)

print_json("footballers.json")

def read_json(path):
    with open(path) as f:
        data = json.load(f)

    return data
```

# Local scope

```
number = 40

def other_number():
    number = 50
    print(number)

other_number()
print(number)
```

```
number = 40

def other_number():
    number = number + 10
    print(number)

other_number()
print(number)
```

```
people = []

def add_person(person):
    people.append(person)

add_person("Barrie")
print(people)
```

## Wikipedia API with functions

Write a terminal program that asks the user for a Wikipedia article and if they want the **description** or **extract**. Then lookup and **return** that data using a function that requires an article title and the value (description or extract). Print the returned data.

- \* Look at the **wikifunction-skeleton.py** file on how to access the Wikipedia API.
- \* Transform the API call to a function that accepts a **title** and **value** argument.
- \* Return different data, depending on what kind of **value** you get.
- \* Ask the user for **title** and **value**.  
Remember to clean up user input using **strip()**.
- \* Call the function and assign the new data to a variable.
- \* Print the returned data

## Tips

- \* Note that a **return** statement can be used together with the **if** statement
- \*

## Extended exercise

- \* Add the option to print the **desktop url** of the article.
- \* Split your function in **two** functions: one that handles the API call, one that handles returning the correct data.

WEB SCRAPING

Met deze ongelooflijke tover-  
kracht ben ik werkelijk  
onoverwinnelijk geworden!



- \* Every site is different
- \* Dynamic websites are difficult
- \* Sites may block you
- \* You need HTML/CSS/JS knowledge

The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The left panel displays the HTML structure of the page, including the DOCTYPE, head, and body sections. The body section contains various scripts and styles. The right panel shows the 'Styles' tab, which lists CSS rules applied to the body element. The rules are categorized into 'element.style', 'body', and 'Inherited from html.no-touch.no-js.mdl-js'. The 'body' rules include width: 100%, min-height: 100%, font-family: Helvetica, Arial, sans-serif, margin: 0, padding: 0, and word-wrap: break-word. The 'Inherited from' rules include color: #000, font-size: 1em, and line-height: 1.4.

```
<!DOCTYPE html>
<html class="no-touch no-js mdl-js">
  <head>...</head>
  <body class="page--" itemscope itemtype="http://schema.org/WebSite"> == $0
    <div class="mdl-layout__container">...</div>
    <link href="https://fonts.googleapis.com/css?family=Roboto+Mono:400,700|Roboto:400,300,500,700,400italic,700italic" rel="stylesheet" type="text/css">
    <script type="text/javascript" async src="https://www.google-analytics.com/analytics.js"></script>
    <script async src="//www.googletagmanager.com/gtm.js?id=GTM-MB3LRF"></script>
    <script src="/static/js/material design lite-bundle.js"></script>
    <script>...</script>
    <!-- Google Tag Manager -->
    <noscript>...</noscript>
    <script>...</script>
    <!-- End Google Tag Manager -->
  </body>
</html>
```

Styles Computed Event Listeners DOM Breakpoints >

Filter :hov .cls +

element.style {

}

body { tools.css:1

width: 100%;

min-height: 100%;

font-family: Helvetica, Arial, sans-serif;

margin: 0;

padding: 0;

word-wrap: break-word;

}

body { user agent stylesheet

display: block;

margin: 8px;

}

Inherited from html.no-touch.no-js.mdl-js

html { tools.css:1

color: #000;

font-size: 1em;

line-height: 1.4;

}

Pseudo ::selection element

::selection { tools.css:1

## Table Of Contents

<a href="#">Beautiful Soup Documentation</a>
▪ <a href="#">Getting help</a>
<a href="#">Quick Start</a>
<a href="#">Installing Beautiful Soup</a>
▪ <a href="#">Problems after installation</a>
▪ <a href="#">Installing a parser</a>
<a href="#">Making the soup</a>
<a href="#">Kinds of objects</a>
▪ <a href="#">Tag</a>
▪ <a href="#">Name</a>
▪ <a href="#">Attributes</a>
▪ <a href="#">Multi-valued attributes</a>
▪ <a href="#">NavigableString</a>
▪ <a href="#">BeautifulSoup</a>
▪ <a href="#">Comments and other special strings</a>
<a href="#">Navigating the tree</a>
▪ <a href="#">Going down</a>
▪ <a href="#">Navigating using tag names</a>
▪ <a href="#">.contents and .children</a>
▪ <a href="#">.descendants</a>

# Beautiful Soup Documentation

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

These instructions illustrate all major features of Beautiful Soup 4, with examples. I show you what the library is good for, how it works, how to use it, how to make it do what you want, and what to do when it violates your expectations.

The examples in this documentation should work the same way in Python 2.7 and Python 3.2.

You might be looking for the documentation for [Beautiful Soup 3](#). If so, you should know that Beautiful Soup 3 is no longer being developed, and that Beautiful Soup 4 is recommended for all new projects. If you want to learn about the differences between Beautiful Soup 3 and Beautiful Soup 4, see [Porting code to BS4](#).

This documentation has been translated into other languages by Beautiful Soup users:

- 这篇文档当然还有中文版。
- このページは日本語で利用できます(外部リンク)
- 이 문서는 한국어 번역도 가능합니다. (외부 링크)





# Japanese Restaurant "Arigato"

## Menu

- Sushi
- Tempura
- Ramen

[Order here!](#)

```
1 <!doctype html>
2 <html>
3 <head>
4     <title>Japanese Restaurant "Arigato"</title>
5 </head>
6 <body>
7     <h1>Japanese Restaurant "Arigato"</h1>
8     <h2>Menu</h2>
9
10    <ul>
11        <li>Sushi</li>
12        <li>Tempura</li>
13        <li>Ramen</li>
14    </ul>
15
16    <a class="order"
17        href="http://www.example.com">
18        Order here!
19    </a>
20 </body>
21 </html>
```

```
1 from bs4 import BeautifulSoup
2
3 with open("restaurant.html") as f:
4     page = BeautifulSoup(f, "lxml")
5
6     title = page.select("title")
7     items = page.select("ul > li")
8     order = page.select(".order")
9
10    print("Title: " + title[0].get_text())
11
12    for item in items:
13        print(item.get_text())
14
15    print("Order url: " + order[0]["href"])
16
```

## **Restaurant scraping**

Scrape a restaurant website and print details.

- \* Find a restaurant website, preferably one that doesn't have a lot of dynamic content using Javascript.
- \* Save a page to a html file
- \* Load the HTML file using **BeautifulSoup**
- \* Get the **<title>** element and at least two other elements from the other page.
- \* Try to get as much data as possible from the site using scraping.

## **Tips**

- \* Check if you have BeautifulSoup by running a Python **console** and executing **from bs4 import BeautifulSoup**
- \* If that doesn't work you need to do **pip install beautifulsoup4** from the **terminal** or the **Anaconda Prompt**

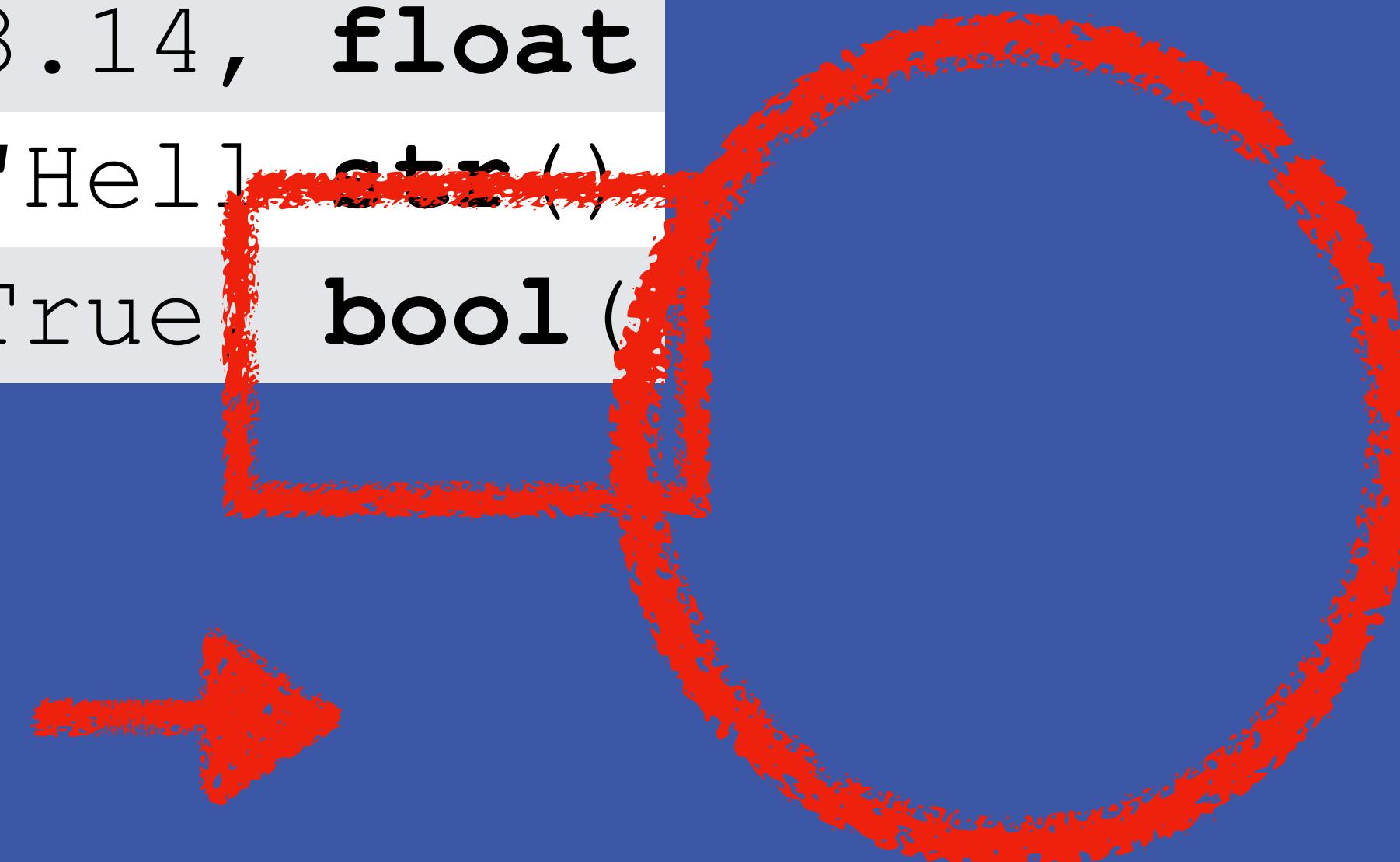
## **Extended exercise**

- \* Save the data to a **CSV** or **JSON** file using **Pandas**



## Type Example Conversion

Integ	42, int()
Float	3.14, float
String	"Hello", str()
Boolean	True, bool()



```
age = 20

if age < 20:
    print("option 1")
elif age <= 20 and age > 20:
    print("option 2")
else:
    print("option 3")
```

Compilation  
Interpretation