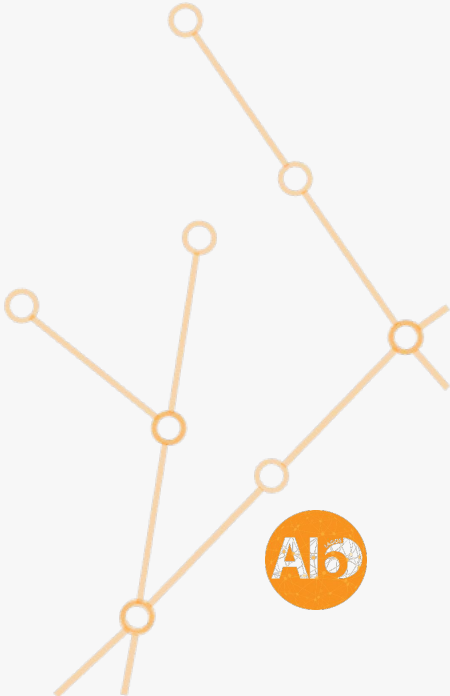
AI SATURDAYS LAGOS

Week 3: Data Collection and Scraping

Instructor: **'Tayo Jabar**

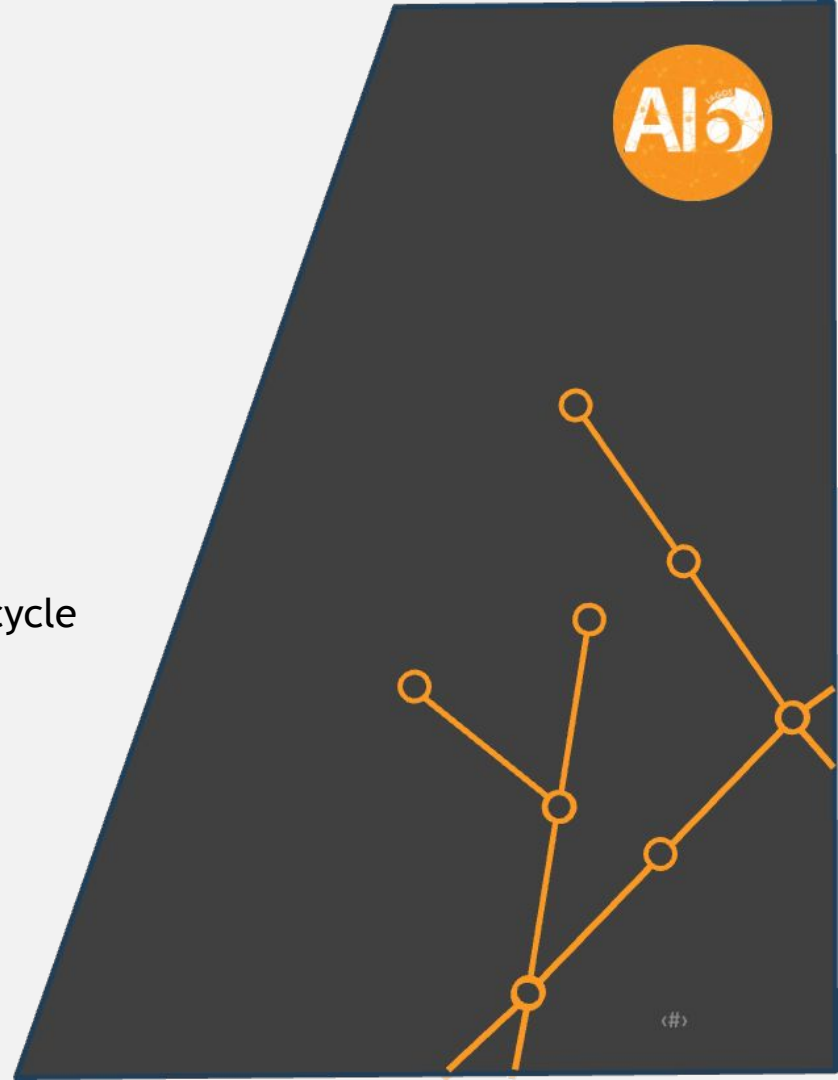
Outline

1. Data Collection
2. Sources of Data
3. Data Formats & Handling
4. Regular Expressions & Parsing



Data Collection

1. Data Collection as part of the data science life cycle



Data Collection as part of a Data Science Life Cycle

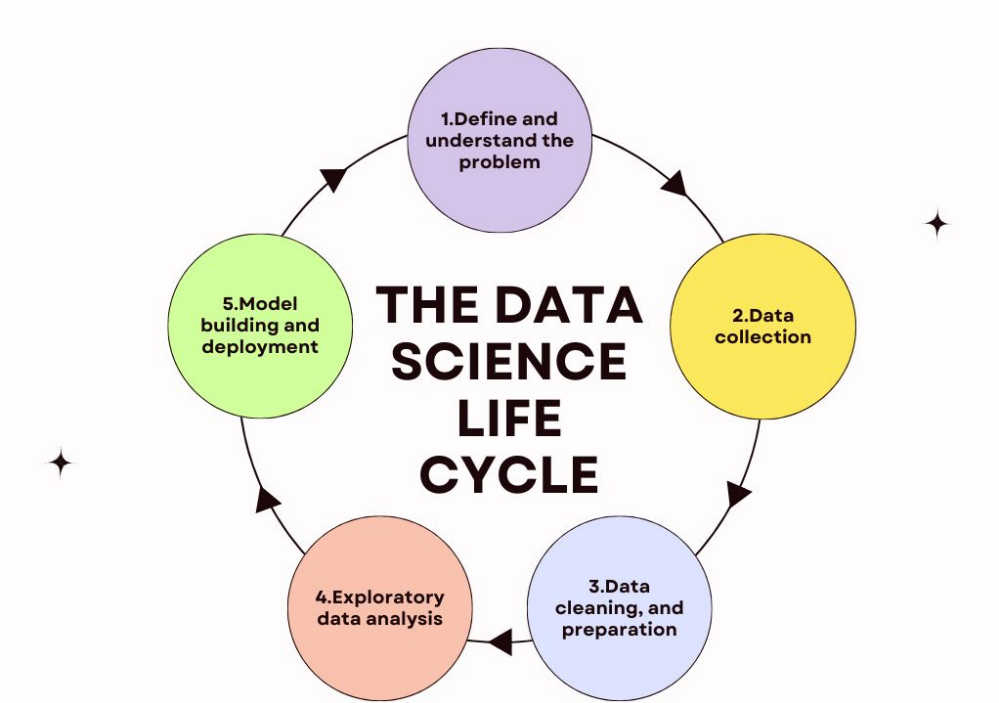
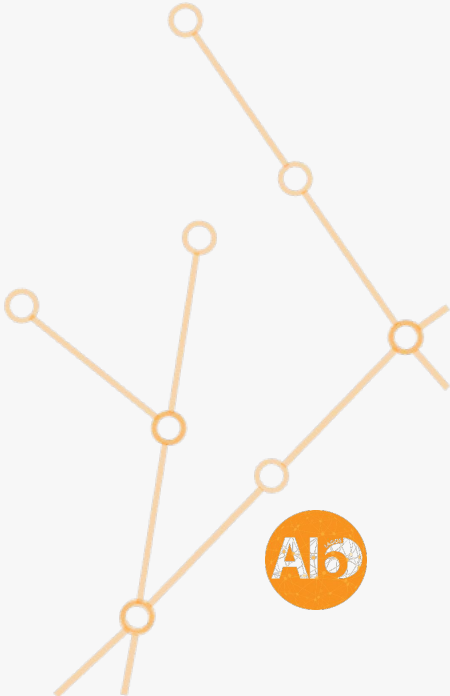


Image Source: Madison Hunter, Towards Data Science

Sources of Data

Sources of Data

1. Surveys, Polls, Focus Groups etc
2. Data extracts or download
3. Database query
4. API queries
5. Web scraping (using HTTP requests)



HTTP Requests

HTTP: HyperText Transfer Protocol

“An HTTP request is made by a client, to a named host, which is located on a server. The aim of the request is to access a resource on the server.” - IBM

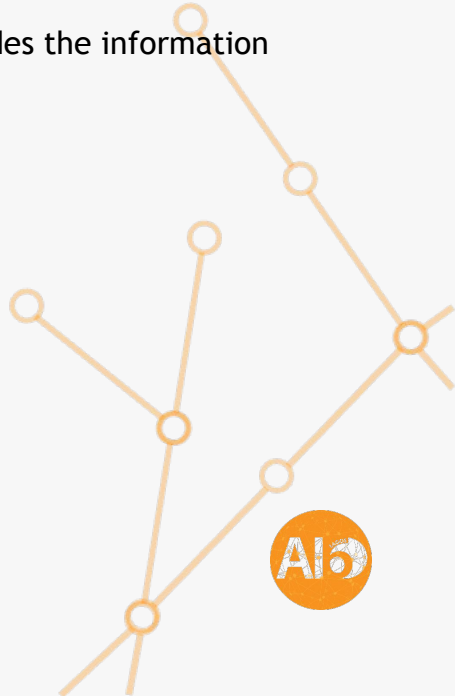
The client uses a URL (Uniform Resource Locator) or in lay terms (web address), which includes the information needed to access the resource. The components of a URL explains URLs.

A complete HTTP request contains at least 2 of the following

- A request line.
- HTTP headers, or header fields.
- A message body, if needed. (Payload)

Standard HTTP Methods:

- GET: To retrieve data from a URL
- POST: To create new data in a resource at the URL
- PUT: To update/replace a resource at a URL
- PATCH: To update and modify a resource at a URL
- DELETE: To delete a resource at a URL



HTTP Requests in Python

Python requests library - http://docs.python-requests.org/

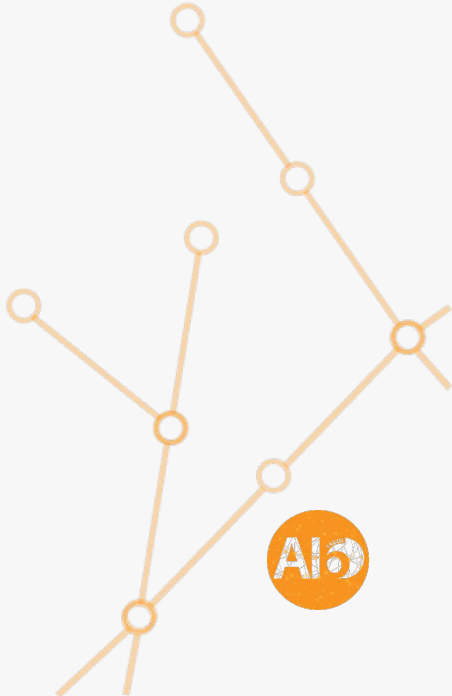
```
import requests

response = requests.get("https://www.google.com")

print (response.status_code)
print (response.content)
print (response.headers)
```

`response.status_code`: 3-digit code that shows the status of a request. Common examples are:

200 - Successful
404 - Not Found
500 - Server Error

`response.content`: This is the actual data contained in the response body.

`response.headers`: This contains metadata about the response received from the server. It contains properties that enable useful manipulation of the response on the client.



HTTP Requests

URL = Address + Query - with/without parameters

```
1  url = "https://www.google.com/search?q=Nexus&oq=Nexus&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDExNjVqMGo3qAIAsAIA&sourceid=chrome&ie=UTF-8"
2
3  # Address: https://www.google.com/search
4  # Parameters: q=Nexus&oq=Nexus&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDExNjVqMGo3qAIAsAIA&sourceid=chrome&ie=UTF-8
5
6  params = {"q": "Nexus", "oq": "Nexus", "gs_lcrp": "EgZjaHJvbWUyBggAEEUYOdIBCDExNjVqMGo3qAIAsAIA", "sourceid": "chrome", "ie":"UTF-8"}
7
8  response = requests.get("https://www.google.com/search", params=params)
9
10 display(response.headers)
```

```
{'Content-Type': 'text/html; charset=ISO-8859-1', 'Date': 'Fri, 25 Aug 2023 19:26:14 GMT', 'Expires': '-1', 'Cache-Control': 'private, max-age=0', 'Content-Security-Policy': "object-src
'self';script-src 'nonce-qa6ui7h5MGXYX_wRWNWgLQ' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/xsrp", 'P3P': '
policy! See g.co/p3phelp for more info."', 'Content-Encoding': 'gzip', 'Server': 'gws', 'X-XSS-Protection': '0', 'X-Frame-Options': 'SAMEORIGIN', 'Set-Cookie': '1P_JAR=2023-08-25-19; exp
19:26:14 GMT; path=/; domain=.google.com; Secure, AEC=Ad49MVHq6jxUTV97G6Tt6BHd6TIrc5iQnIrtkPjJKlzoR6beAa1LdC_d7EM; expires=Wed, 21-Feb-2024 19:26:14 GMT; path=/; domain=.google.com; Secu
NID=511=EpzwYDzpehuedcR_ZRM0jEwNWsV2I_Dwko6YaNUcdxOdQgGO8FiJwS8gTKO4aC66T3an8l4n8aix_vI8ayEllV4fG5Fk-Qkn1RPNC88MuVcFm4w8oBauTb7PtT7RARSy84I7fHdP5S_OIMPoVpWsiOUSK6Rs-kK-kPPE6IbW-NU; expire
GMT; path=/; domain=.google.com; HttpOnly', 'Transfer-Encoding': 'chunked'}
```

API Queries (REST(ful) APIs)

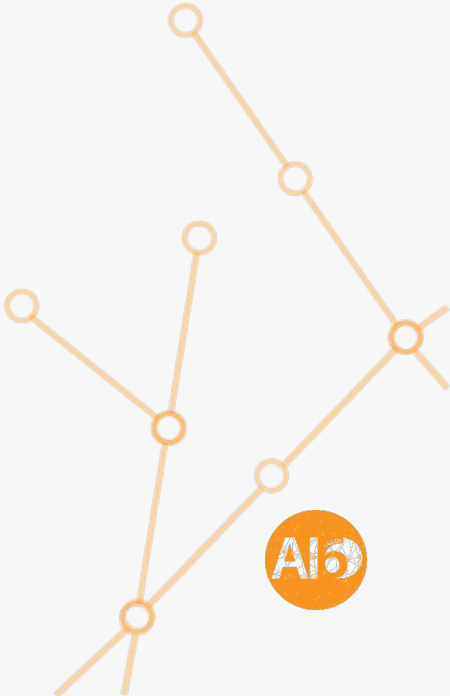
REST stands for REpresentational State Transfer

API stands for Application Programming Interface

A web API that conforms to the REST architectural design is considered RESTful.

The principles of the REST architecture:

- Uniform Interface
- Client - Server
- Stateless
- Cacheable
- Layered System
- Code on demand(optional)



Interacting with RESTful APIs

- Standard HTTP methods: GET, POST, PUT, DELETE
- Parameters and headers usually need to be specified (Authentication)

```
my_header = {'Authorization' : 'Bearer {token}'} ## Generate your own token from github at https://github.com/settings/tokens/new
response = requests.get("https://api.github.com/user", headers=my_header)

# print(response.status_code)
# print(response.headers["Content-Type"])
# print(response.json().keys())
print(response.content)
```

- Authorization(OAuth) Vs Authentication (Auth)
  - Authentication - Verification (Who are you?) e.g username and password logins
  - Authorization - Access Permission (What level of access do you have?) e.g access tokens

```
response = requests.get("https://api.github.com/user", auth=("tayojabar", "{password}"))
# print(response.status_code)
# print(response.headers["Content-Type"])
# print(response.json().keys())
print(response.content)
```

Quiz

Question A: *Question: Which HTTP method is typically used to retrieve data from a server without modifying it?*

A) GET

B) POST

C) PUT

D) DELETE

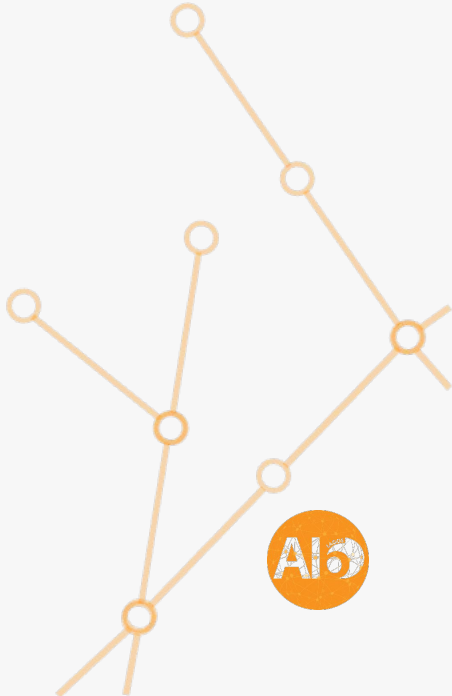
Question B: *In the context of RESTful APIs, what does "REST" stand for?*

A) Real-time Extraction and Server Transfer

B) Remote Environment for Server Testing

C) Representational State Transfer

D) Responsive Entity Schema Technology



Quiz

Question C: Which authentication method involves including a token in the request header to access protected resources in a RESTful API?

A) Basic Authentication

B) OAuth

C) API Key Authentication

D) Digest Authentication

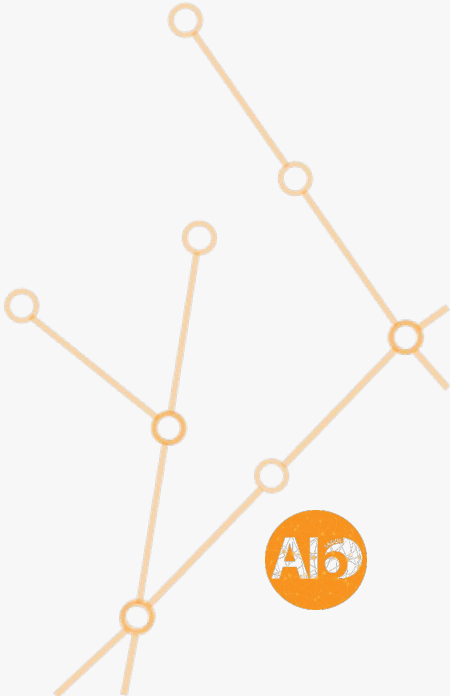
Question D: In a RESTful API, what does the term "resource" refer to?

A) The URL of the API

B) The authentication token

C) A piece of data that the API can provide access to

D) The HTTP request method used



Data Formats & Handling

1. CSV - Comma Separated Values
2. JSON - JavaScript Object Notation
3. HTML/XML - HyperText Markup Language / Extensible Markup Language

CSV Files

- Sample_file.csv
- Text files with some form of delimiter (comma, tab)

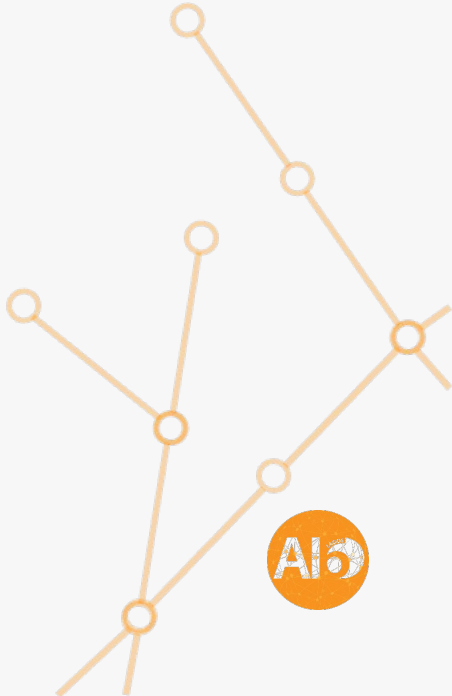
```
ZTime,Time,OAT,DT,SLP,WD,WS,SKY,PPT,PPT6,Plsr.Event,Plsr.Source
20170820040000,20170820000000,178,172,10171,0,0,0,0,-9999,,
20170820050000,20170820010000,178,172,10177,0,0,0,0,-9999,,
20170820060000,20170820020000,167,161,10181,0,0,0,0,-9999,,
20170820070000,20170820030000,161,161,10182,0,0,4,0,-9999,,
20170820080000,20170820040000,156,156,10186,180,15,-9999,0,-9999,,
20170820090000,20170820050000,161,161,10192,0,0,9,0,-9999,,
20170820100000,20170820060000,156,156,10196,0,0,9,0,-9999,,
20170820110000,20170820070000,150,150,-9999,0,0,-9999,0,-9999,,
```

- Common issues with reading CSV files include:
  - Values containing commas
  - Encoding format
  - Parsing date values

```
import pandas as pd

dataframe = pd.read_csv("kpit_weather.csv", delimiter=",", quotechar='"', encoding="utf-8", parse_dates=['date'])
dataframe.head()
```

JSON Data

- Adapted from JavaScript as a method of encapsulating objects
- A lightweight data-interchange format i.e adaptable and readable in various programming languages eg Python, Java, C, JavaScript etc.
- Typically appears in name/value pairs
- Values of different data types can be represented in JSON format
    - Numbers, Strings, Boolean, Arrays, Dictionaries etc



JSON data (...Cont'd)

```
{
  "name": "John Doe",
  "age": 30,
  "email": "john@example.com",
  "address": {
    "street": "123 Main Street",
    "city": "Cityville",
    "state": "State",
    "postal_code": "12345"
  },
  "interests": ["reading", "hiking", "photography"],
  "is_student": false
}
```

JSON Handling in Python

- Python's built-in library for handling JSON data and files
  - https://docs.python.org/3/library/json.html

```
import json
import requests

response = requests.get("https://api.github.com/user", auth=("tayojabar", "{password}"))

#loads, load, dumps, dump
info = json.loads(response.content)
info_string = json.dumps(info)

display (info)
display (info_string)
```

HTML/XML Data

- The web is mostly filled with XML tagged data
- Text is encoded in a structure of hierarchical tags.

```
<tag attribute="value">
    <subtag>
        Some content for the subtag
    </subtag>
    <openclosetag attribute="value2”/>
</tag>
```

- HTML on the other hand, though largely similar to XML is considered difficult to parse due to the asymmetry in some of the tags i.e some open tags do not get closed.
- HTML was built majorly to *display* data and not to structure data.
- However, most of the web pages you’ll encounter today are in HTML as it is now the standard.

Parsing HTML/XML in Python

- There are a couple of libraries that exist in Python for parsing XML, e.g. ElementTree, Minidom
- One that parses both HTML and XML equally well is BeautifulSoup - https://pypi.org/project/beautifulsoup4/

```
import requests
from bs4 import BeautifulSoup

url = 'https://example.com'
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')

# Extracting data from the parsed HTML
title = soup.title.text
paragraphs = soup.find_all('p')

print("Title:", title)
for p in paragraphs:
    print("Paragraph:", p.text)
```

✓ 2.0s

```
Title: Example Domain
Paragraph: This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.
Paragraph: More information...
```

Quiz

Question A: One of the following is the correct meaning of the JSON data format

A) JavaScript Object Notation

B) Java Server Object Notation

C) JavaScript Object Nodes

D) Just Send On Notepad

Question B: Which Python library is commonly used for parsing HTML content during web scraping?

A) PyScrape

B) BeautifulRequests

C) BeautifulSoup

D) ScrapyParser

We Value Your Feedback.

https://docs.google.com/forms/d/e/1FAIpQLSfoYXRTHJfjQo1Xj2PrgoNLKWOAUeiVl4MLrRrOACtYFzM8cg/viewform?usp=pp_url

Regular Expressions

1. Finding & Matching single strings
2. Matching multiple strings
3. Grouping
4. Substitution

Finding & Matching singular strings

- The Python `re` library - https://docs.python.org/3/library/re.html

```
import re
text = "This course will introduce the basics of data science"
match = re.search(r"data science", text)
print (match)
print(match.start())
```

```
<re.Match object; span=(41, 53), match='data science'>
41
```

Other functions:

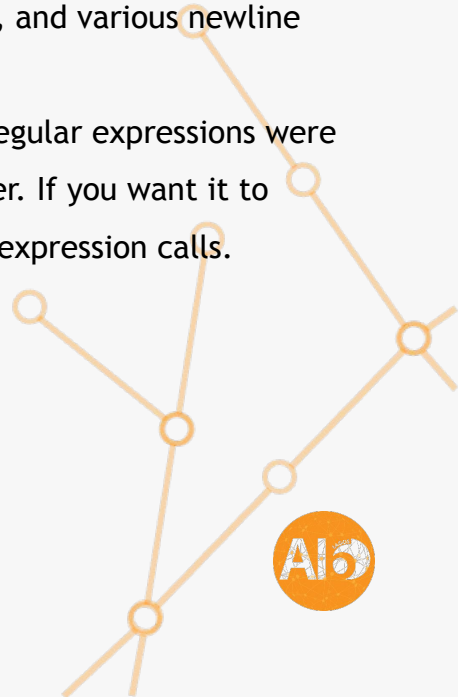
- `re.match()`: Match the regular expression starting at the beginning of the text string
- `re.finditer()`: Find all matches in the text, returning a iterator over match objects
- `re.findall()`: Find all matches in the text, returning a list of the matched text only (not a match object)

Matching Multiple Strings

- Any character (except special characters, `".$*+?{}\[]|()`), just matches itself. I.e., the character `a` just matches the character `a`. This is actually what we used previously, where each character in the `r"data science"` regular expression was just looking to match that exact character.
- Putting a group of characters within brackets `[abc]` will match any of the characters a, b, or c. You can also use ranges within these brackets, so that `[a-z]` matches any lower case letter.
- Putting a caret(`^`) within the bracket matches anything but these characters, i.e., `[^abc]` matches any character except `a`, `b`, or `c`.

Matching Multiple Strings

- The special character `\d` will match any digit, i.e. `[0-9]`
- The special character `\w` will match any alphanumeric character plus the underscore; i.e., it is equivalent to `[a-zA-Z0-9_]`.
- The special character `\s` will match whitespace, any of `[ \t\n\r\f\v]` (a space, tab, and various newline characters).
- The special character `.` (the period) matches any character. In their original versions, regular expressions were often applied line-by-line to a file, so by default, `.` will not match the newline character. If you want it to match newlines, you pass `re.DOTALL` to the "`flags`" argument of the various regular expression calls.



Grouping

- `group()` function: Returns all groups present in the text as a tuple

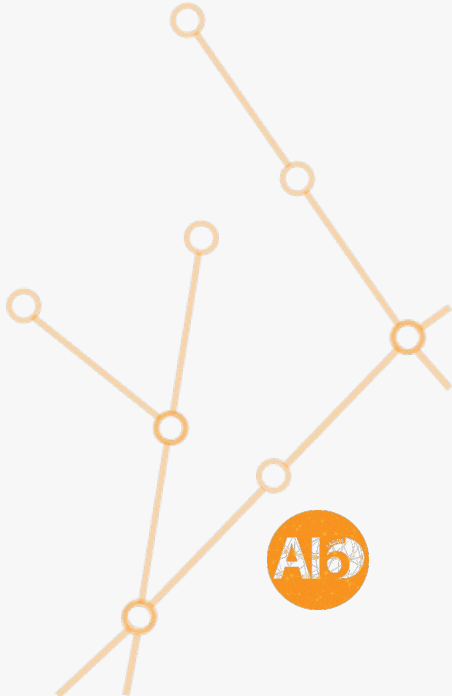
```
match = re.search(r"(\w+)\s([Ss]cience)", text)
print(match.groups())
```

```
('Data', 'Science')
```

- `group(i)` function: Returns the groups present in ith position of the groups tuple, with the notation `group(0)` returning the entire text itself

```
match = re.search(r"(\w+)\s([Ss]cience)", text)
print(match.group(0))
print(match.group(1))
print(match.group(2))
```

```
Data Science
Data
Science
```



Substitution

- `sub(a, b, c)` function: Replaces regular expression `a` with `b` in original text `c`

```
print(re.sub(r"data science", r"data schmience", text))
```

✓ 0.0s

```
This course will introduce the basics of data schmience
```

Quiz

Question A: *What does the metacharacter '*' signify in regular expressions?*

A) Match the start of a line

B) Match the end of a line

C) Match zero or more occurrences of the preceding element

D) Match one or more occurrences of the preceding element

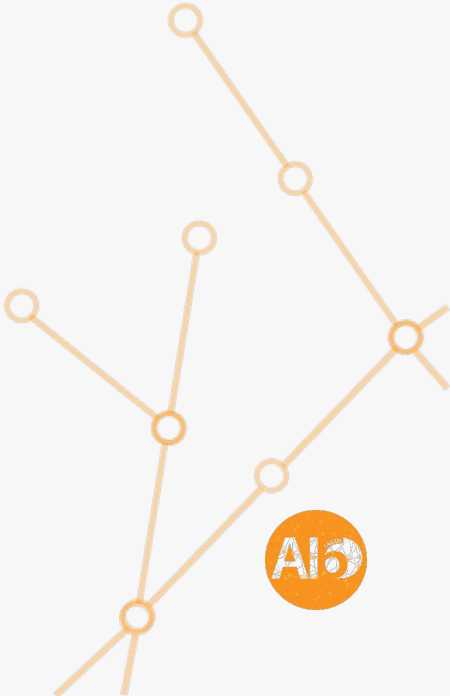
Question B: *What does the metacharacter '[...]' signify in regular expressions?*

A) Match the start of a line

B) Match any character within the brackets

C) Match none of the characters within the brackets

D) Creates a new regular expression



Thank You

Resources and External Links

- Practical Data Science: Data Collection and Scraping - CMU AI, Zico Kolter, Pat Virtue
  - https://www.datasciencecourse.org/notes/data_collection/
  - 15-388/688 - Practical Data Science: Data collection and web scraping
- HTTP Requests Documentation - IBM, HTTP requests
- JSON, https://www.json.org/json-en.html
- Regular Expressions Cheat Sheet - DataCamp, https://www.datacamp.com/cheat-sheet/regular-expresso