https://bmtgoncalves.github.io/TorinoCourse/

Lecture I - Web Scraping

Bruno Gonçalves

www.bgoncalves.com

NYU

DATA SCIENCE

Requirements

https://bmtgoncalves.github.io/TorinoCourse/

ANACONDA

python

3.5

PostgreSQL

QGIS 2.18

Las Palmas de G.C.

PostGIS

Spatial PostgreSQL

urllib

https://docs.python.org/3/library/urllib.html

- Extensible library for opening and manipulating **URLs**
- https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f?s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw
    - https <- protocol
    - foursquare.com <- server
    - /tyayayayaa/checkin/5304b652498e734439d8711f <- resource within server
    - s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw <- Query string

urllib

https://docs.python.org/3/library/urllib.html

- Extensible library for opening and manipulating **URLs**
- https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f?s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw
    - https <- protocol
    - foursquare.com <- server
    - /tyayayayaa/checkin/5304b652498e734439d8711f <- resource within server
    - s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw <- Query string

```python
from urllib import parse

url = "https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f?s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw"

parsed = parse.urlparse(url)
query = parsed.query
query_dict = parse.parse_qs(query)

print(parsed)
print(query_dict)
```

urllib_parse.py

urllib

https://docs.python.org/3/library/urllib.html

- urllib2.urlopen(url) opens a url for reading and returns a "file handle"-like object
- Information about the webpage can be obtained with the .info() method in the form of an HTTPMessage
- The HTTPMessage object obeys the usual Python dictionary interface
- The .geturl() method returns the final location of the webpage.
  .urlopen() follows redirects until it connects with the final content.
- .getcode() returns the status code of the call
  - 200 OK
  - 404 File Not Found
  - 500 Internal Server Error

Challenge - urllib

https://docs.python.org/3/library/urllib.html

- Find the final location of the shortened url:

http://bit.ly/GoogleScholar

- and access the headers and info of the request

```
from urllib import request

url = "http://bit.ly/GoogleScholar"

webpage = request.urlopen(url)
code = webpage.getcode()
info = webpage.info()

headers  = info

new_url = webpage.geturl()

print(url, "redirected to", new_url)
```

@bgoncalves

urllib_request.py

posixpath

http://docs.python.org/3/library/os.path.html

- Manipulate paths in a POSIX operating system
- Also useful to extract information from remote resource paths
- Aliased to os.path if your operating systems is POSIX
- https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f
  -> Path in remote filesystem
- .basename(path) -> returns the file name (if there is one) 5304b652498e734439d8711f
- .dirname(path) -> return the directory portion /tyayayayaa/checkin

posixpath

http://docs.python.org/3/library/os.path.html

```python
from urllib import parse
import posixpath

url = "https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f?s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw"

parsed = parse.urlparse(url)
filename = posixpath.basename(parsed.path)
directory = posixpath.dirname(parsed.path)

print(filename, directory)
```

@bgoncalves

posixpath_demo.py

requests

http://requests.readthedocs.org/en/latest/

- "HTTP for Humans" - Simplified HTTP requests:
  - authentication (basic authentication, OAuth1, OAuth2, etc) [See next lecture]
  - header manipulation
  - error handling
  - etc…

requests

http://requests.readthedocs.org/en/latest/

- .get(url) open the given url for reading and returns a Response
- Response.status_code is a field that contains the calls status code
- Response.headers is a dict containing all the returned headers
- Response.text is a field that contains the content of the returned page
- Response.url contains the final url after all redirections
- Response.json() parses a JSON response (throws a JSONDecodeError exception if response is not valid JSON). Check "content-type" header field.

requests

http://requests.readthedocs.org/en/latest/

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

url = "http://bit.ly/GoogleScholar"

req = requests.get(url)
print("Status code:", req.status_code)
print("Server Header Information:")
pprint(req.headers)

new_url = req.url

print(url, "redirected to", new_url)
```

requests_demo.py

json

https://docs.python.org/3/library/json.html

- JavaScript Object Notation - Serialization format originally developed for Javascript
- Currently widely accepted format for data dissemination
- Most languages have excellent libraries to handle it
- json.loads(obj_str) - load JSON data from a string - returns native Python object
- json.load(fp) - load JSON data from a file handle - returns native Python object
- json.dumps(obj) - convert JSON data to a string
- json.dump(obj, fp) - write the string version of obj to the file handle fp

Challenge - json

https://docs.python.org/3/library/json.html

- Access the JSON file:

http://www.bgoncalves.com/test.json

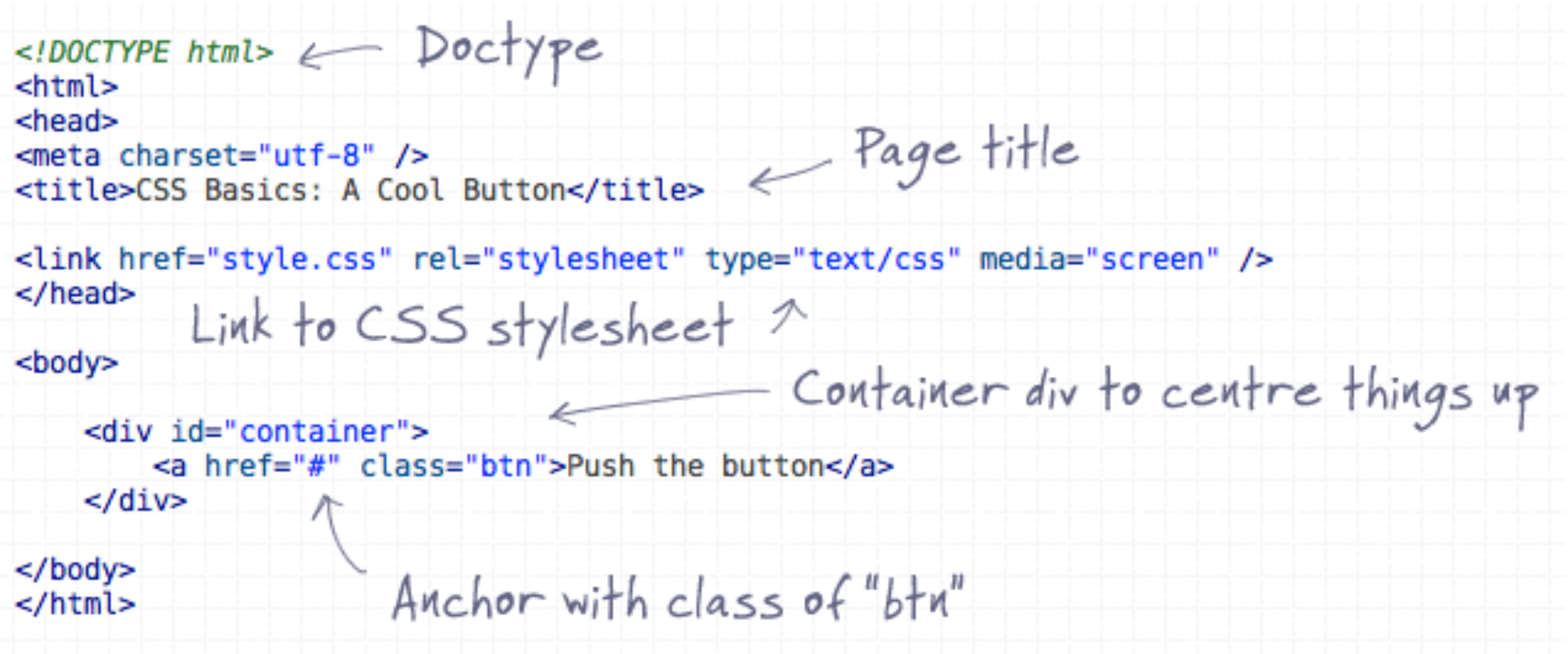
- and extract all the friend pairs

```
import requests
import json

url = "http://www.bgoncalves.com/test.json"

request = requests.get(url)

data = json.loads(request.text)

for user in data:
    name = user["name"]

    for friend in user["friends"]:
        print(name, "->", friend["name"])
```

json_demo.py

Basic Structure of a web page

http://www.w3schools.com/tags/

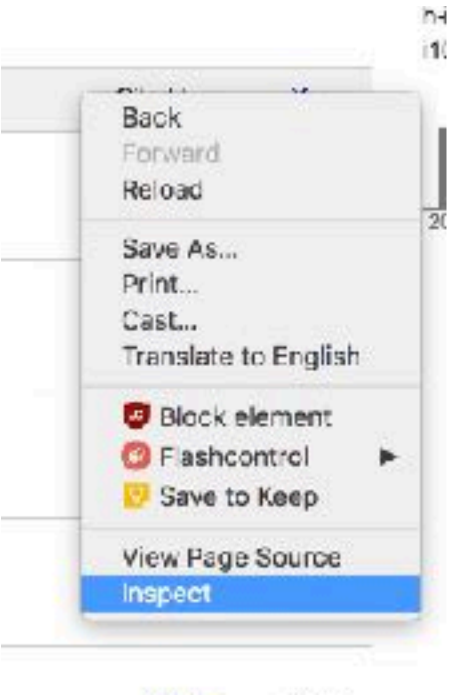


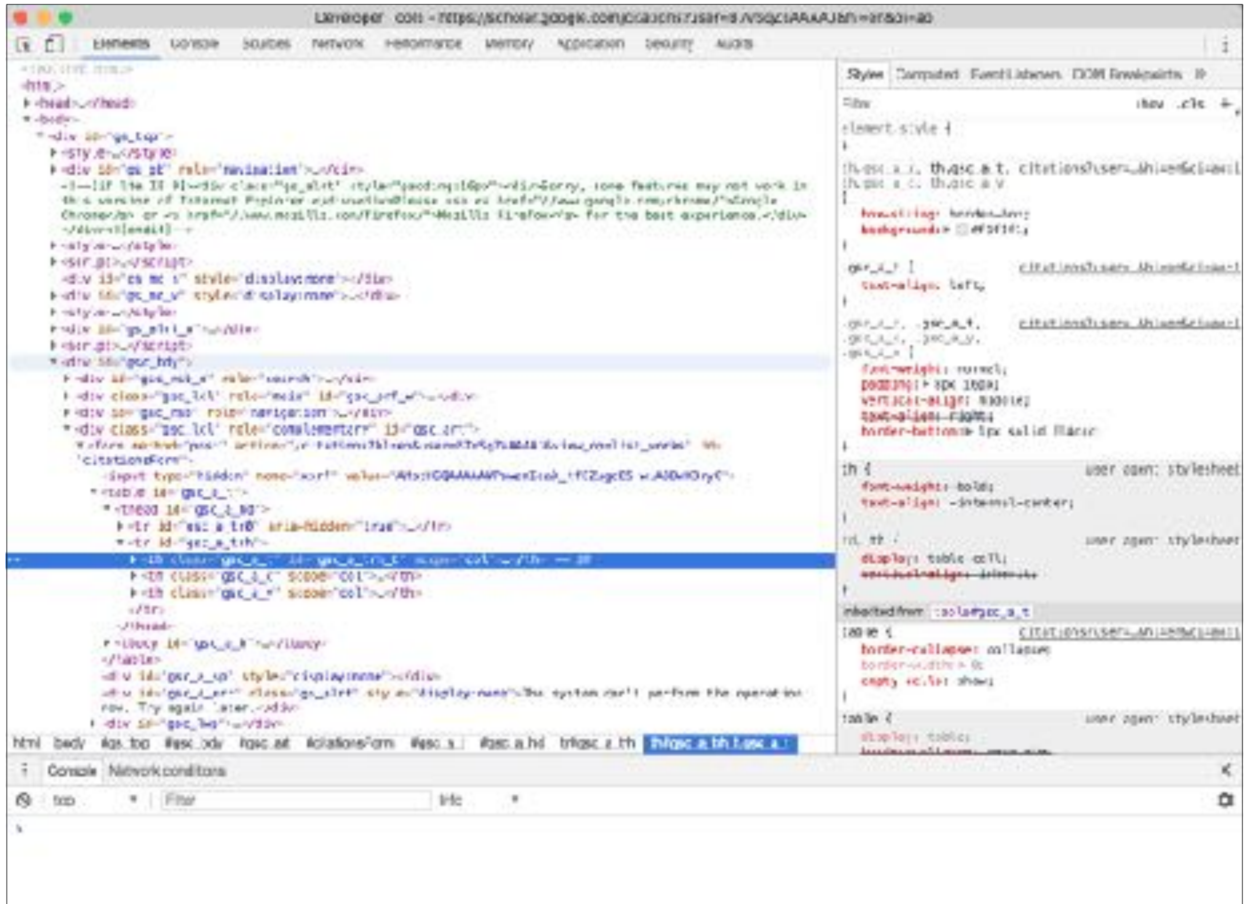
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>CSS Basics: A Cool Button</title>

<link href="style.css" rel="stylesheet" type="text/css" media="screen" />
</head>

<body>

    <div id="container">
        <a href="#" class="btn">Push the button</a>
    </div>

</body>
</html>
```

- Tree-like structure (DOM)
- Nested <tags> with attributes and content
- Two main sections under <html>:
    - <head> - meta data and resource location
    - <body> - page contents

Chrome Developer Tools

- Extremely powerful and intuitive set of tools
- Comes standard with Google Chrome. Just right click anywhere on the page and select "Inspect"
- Allows you to interactively change the DOM of any "live" webpage and find which element corresponds to which part of the page.





Demo

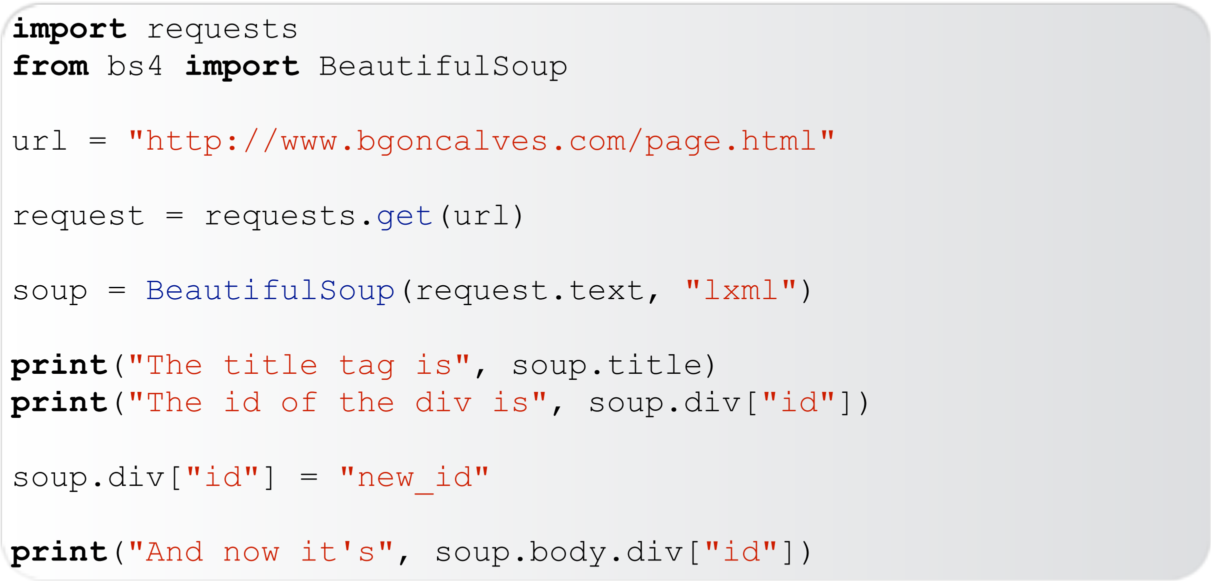
BeautifulSoup

http://www.crummy.com/software/BeautifulSoup/bs4/doc/

- Parses html and xml files into a tree.
- BeautifulSoup(page) where page is a string or a “file handle”-like object
- BeautifulSoup parses the contents of the page and returns a BeautifulSoup object, corresponding to the root of the document tree.
- Each leaf of the tree is a Tag object:
    - can be used as dicts to access tag attributes,
    - contains pointers to children (.findChildren()), siblings (.findSiblings()) and parent (.findParent())
    - can be accessed recursively by name (head.title.content)
    - modifying the contents of a tag modifies the contents of the document

BeautifulSoup

http://www.crummy.com/software/BeautifulSoup/bs4/doc/



```
import requests
from bs4 import BeautifulSoup

url = "http://www.bgoncalves.com/page.html"

request = requests.get(url)

soup = BeautifulSoup(request.text, "lxml")

print("The title tag is", soup.title)
print("The id of the div is", soup.div["id"])

soup.div["id"] = "new_id"

print("And now it's", soup.body.div["id"])
```

- .findAll() returns a list of all tags matching a certain criteria
  - .findAll(name=“a”) find all “a” tags (links)
  - .findAll(name=[“a”, “div”]) find all “a” and “div” tags
  - .findAll(attrs = {"class": “btn"}) find all tags with class “btn”, regardless of tag name
  - .findAll(name=“a”, attrs = {"class": “btn”}, limit=2) find the first two “a” tags with class “btn”

soup.py

BeautifulSoup

http://www.crummy.com/software/BeautifulSoup/bs4/doc/

```
import requests
from bs4 import BeautifulSoup

url = "http://www.whoishostingthis.com/tools/user-agent/"

headers = {"User-agent" : "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:25.0) Gecko/20100101 
Firefox/25.0"}

request_default = requests.get(url)
request_spoofed = requests.get(url, headers=headers)

soup_default = BeautifulSoup(request_default.text, "lxml")
soup_spoofed = BeautifulSoup(request_spoofed.text, "lxml")

print("Default:", soup_default.find(name="div", attrs={"class":"info-box user-agent"}).text)
print("Spoofed:", soup_spoofed.find(name="div", attrs={"class":"info-box user-agent"}).text)
```

requests_spoof.py

BeautifulSoup

http://www.crummy.com/software/BeautifulSoup/bs4/doc/

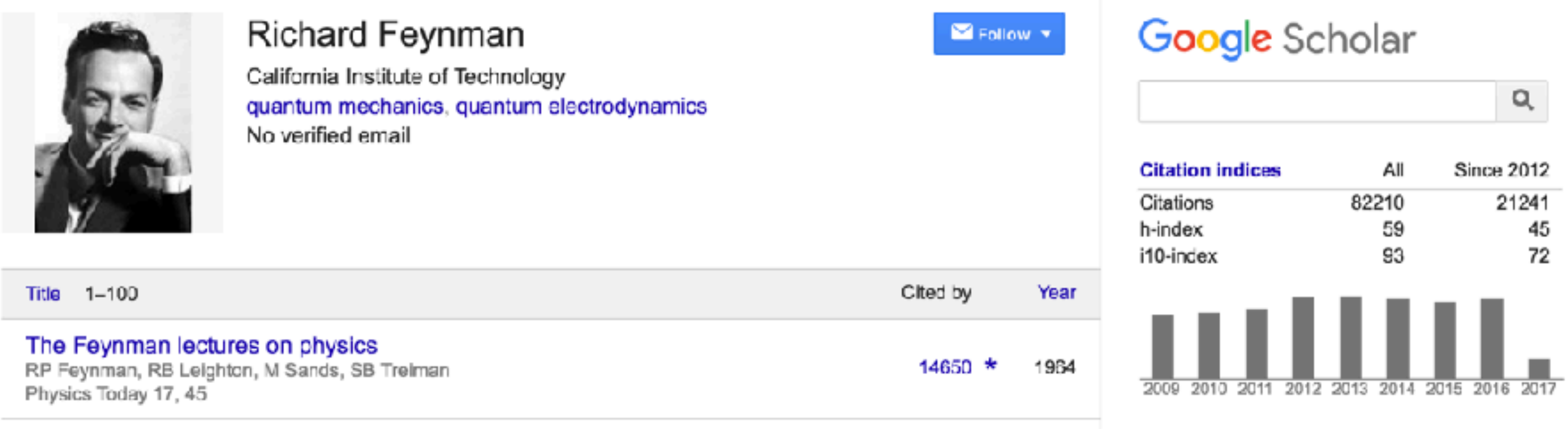
```python
import requests
from bs4 import BeautifulSoup

url = "http://www.whoishostingthis.com/tools/user-agent/"

headers = {"User-agent" : "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:25.0) Gecko/20100101 
Firefox/25.0"}

request_default = requests.get(url)
request_spoofed = requests.get(url, headers=headers)

soup_default = BeautifulSoup(request_default.text, "lxml")
soup_spoofed = BeautifulSoup(request_spoofed.text, "lxml")

print("Default:", soup_default.find(name="div", attrs={"class":"info-box user-agent"}).text)
print("Spoofed:", soup_spoofed.find(name="div", attrs={"class":"info-box user-agent"}).text)
```

- Some servers use the User-agent string to decide how to format the output
  - Correctly handle specific versions of web browsers
  - Provide lighter/simplified versions to users on their mobiles
  - Refusing access to automated tools, etc

requests_spoof.py

Challenge - BeautifulSoup

- Extract the title of Feynman's 100 most cited papers from Google Scholar



Richard Feynman

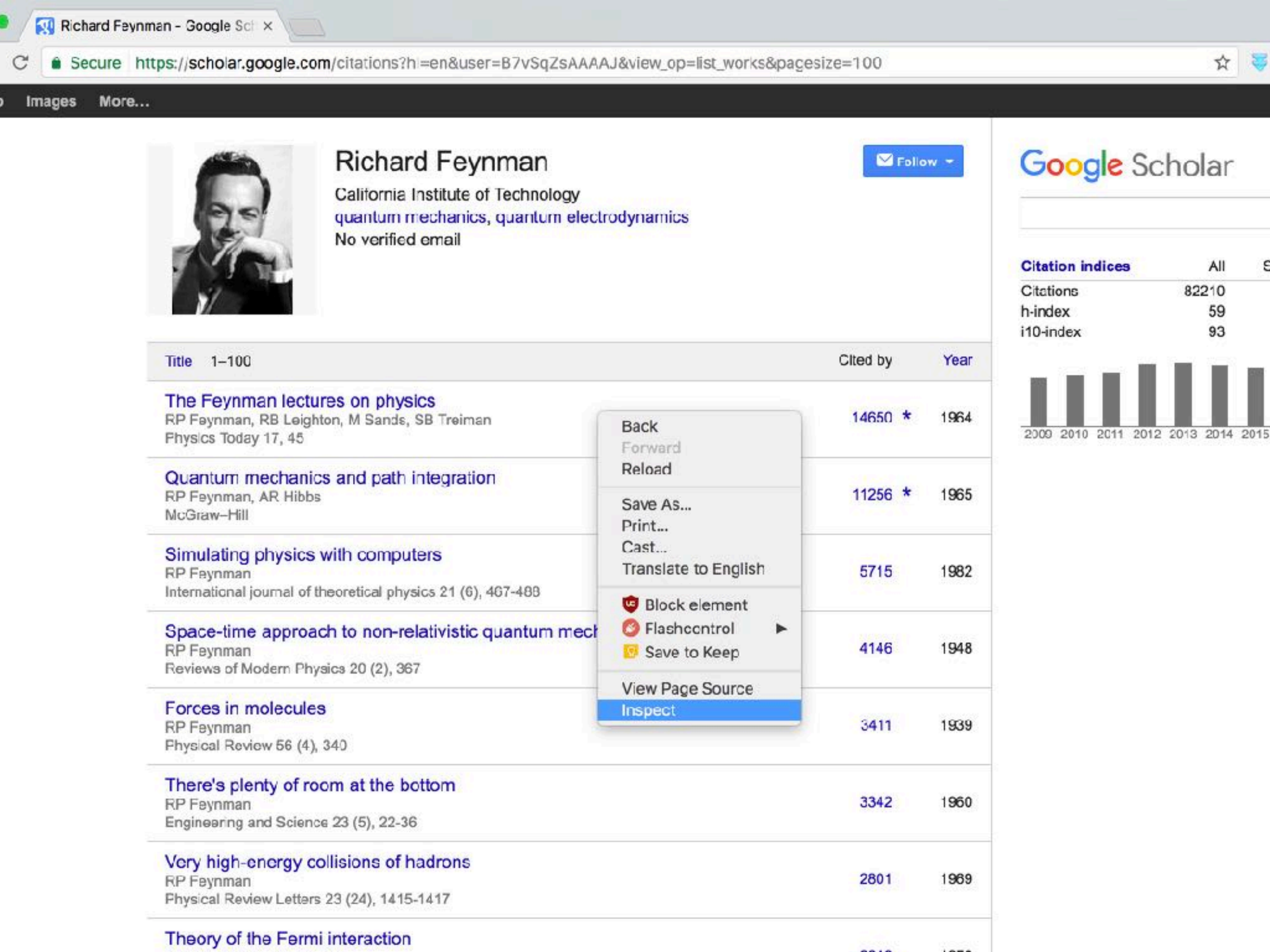
California Institute of Technology

quantum mechanics, quantum electrodynamics

No verified email

Title 1–100	Cited by	Year
The Feynman lectures on physics RP Feynman, RB Leighton, M Sands, SB Treiman Physics Today 17, 45	14650 *	1964

Citation indices	All	Since 2012
Citations	82210	21241
h-index	59	45
i10-index	93	72



https://scholar.google.com/citations?hl=en&user=B7vSqZsAAAAJ&view_op=list_works&pagesize=100

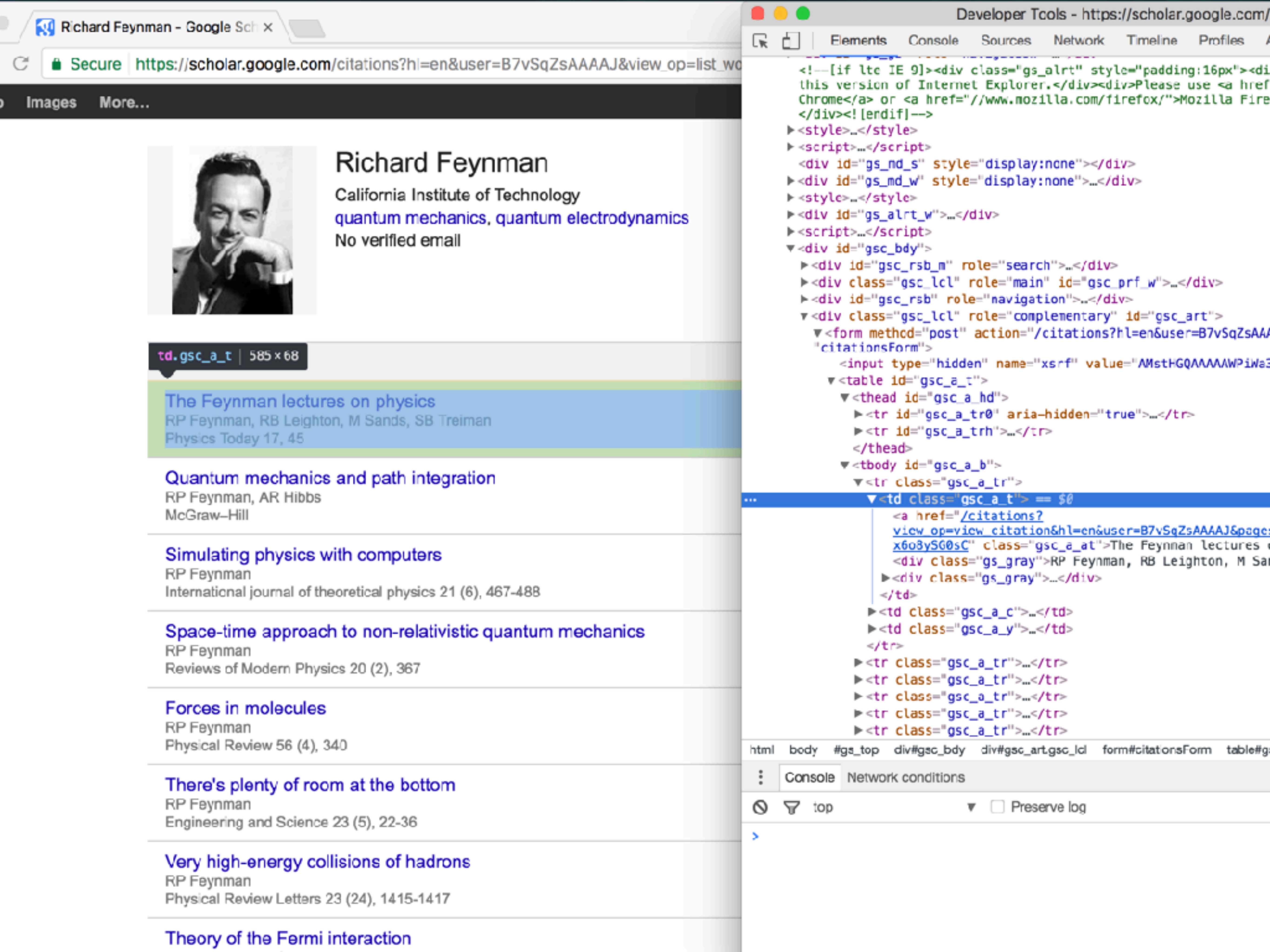
Richard Feynman

California Institute of Technology
quantum mechanics, quantum electrodynamics
No verified email

Google Scholar

Citation indices	All
Citations	82210
h-index	59
i10-index	93

Title 1–100	Cited by	Year
The Feynman lectures on physics RP Feynman, RB Leighton, M Sands, SB Treiman Physics Today 17, 45	14650 *	1964
Quantum mechanics and path integration RP Feynman, AR Hibbs McGraw-Hill	11256 *	1965
Simulating physics with computers RP Feynman International journal of theoretical physics 21 (6), 467-488	5715	1982
Space-time approach to non-relativistic quantum mech RP Feynman Reviews of Modern Physics 20 (2), 367	4146	1948
Forces in molecules RP Feynman Physical Review 56 (4), 340	3411	1939
There's plenty of room at the bottom RP Feynman Engineering and Science 23 (5), 22-36	3342	1960
Very high-energy collisions of hadrons RP Feynman Physical Review Letters 23 (24), 1415-1417	2801	1969
Theory of the Fermi interaction		



Richard Feynman - Google Sch ×
Secure
https://scholar.google.com/citations?hl=en&user=B7vSqZsAAAAJ&view_op=list_wo
Images
More...

Richard Feynman

California Institute of Technology

quantum mechanics, quantum electrodynamics

No verified email

The Feynman lectures on physics
RP Feynman, RB Leighton, M Sands, SB Treiman
Physics Today 17, 45

Quantum mechanics and path integration
RP Feynman, AR Hibbs
McGraw-Hill

Simulating physics with computers
RP Feynman
International journal of theoretical physics 21 (6), 467-488

Space-time approach to non-relativistic quantum mechanics
RP Feynman
Reviews of Modern Physics 20 (2), 367

Forces in molecules
RP Feynman
Physical Review 56 (4), 340

There's plenty of room at the bottom
RP Feynman
Engineering and Science 23 (5), 22-36

Very high-energy collisions of hadrons
RP Feynman
Physical Review Letters 23 (24), 1415-1417

Theory of the Fermi interaction

```
  <!--[if lte IE 9]><div class="gs_alrt" style="padding:16px"><di
  this version of Internet Explorer.</div><div>Please use <a href
  Chrome</a> or <a href="//www.mozilla.com/firefox/">Mozilla Fire
  </div><![endif]-->
▶<style>…</style>
▶<script>…</script>
  <div id="gs_md_s" style="display:none"></div>
▶<div id="gs_md_w" style="display:none">…</div>
▶<style>…</style>
▶<div id="gs_alrt_w">…</div>
▶<script>…</script>
▼<div id="gsc_bdy">
  ▶<div id="gsc_rsb_m" role="search">…</div>
  ▶<div class="gsc_lcl" role="main" id="gsc_prf_w">…</div>
  ▶<div id="gsc_rsb" role="navigation">…</div>
  ▼<div class="gsc_lcl" role="complementary" id="gsc_art">
    ▼<form method="post" action="/citations?hl=en&user=B7vSqZsAAA
    "citationsForm">
        <input type="hidden" name="xsrf" value="AMstHGQAAAAAWPiWa3
      ▼<table id="gsc_a_t">
        ▼<thead id="gsc_a_hd">
          ▶<tr id="gsc_a_tr0" aria-hidden="true">…</tr>
          ▶<tr id="gsc_a_trh">…</tr>
          </thead>
        ▼<tbody id="gsc_a_b">
          ▼<tr class="gsc_a_tr">
...         ▼<td class="gsc_a_t"> == $0
                <a href="/citations?
                view_op=view_citation&hl=en&user=B7vSqZsAAAAJ&page
                x6o8ySG0sC" class="gsc_a_at">The Feynman lectures o
                <div class="gs_gray">RP Feynman, RB Leighton, M Sa
              ▶<div class="gs_gray">…</div>
              </td>
            ▶<td class="gsc_a_c">…</td>
            ▶<td class="gsc_a_y">…</td>
            </tr>
          ▶<tr class="gsc_a_tr">…</tr>
          ▶<tr class="gsc_a_tr">…</tr>
          ▶<tr class="gsc_a_tr">…</tr>
          ▶<tr class="gsc_a_tr">…</tr>
          ▶<tr class="gsc_a_tr">…</tr>
```

html body #gs_top div#gsc_bdy div#gsc_art.gsc_lcl form#citationsForm table#g
Console
Network conditions
top
Preserve log

Challenge - BeautifulSoup

- Extract the title of Feynman's 100 most cited papers from Google Scholar

```
import requests
from bs4 import BeautifulSoup

url = "http://scholar.google.com/citations?hl=en&user=B7vSqZsAAAAJ&view_op=list_works&pagesize=100"

request = requests.get(url)
soup = BeautifulSoup(request.text, 'lxml')

table = soup.find("table", attrs={"id" : "gsc_a_t"})

for paper in table.findAll("td", attrs={"class": "gsc_a_t"}):
    print(paper.a.string)
```

feynman.py

pyquery

https://pyquery.readthedocs.io/en/latest/
See also: *https://www.w3schools.com/jquery/*

- Python version of the popular jQuery javascript package.
- More powerful than BeautifulSoup but also more complex.
- It defines three type of selectors:
  - element selector - retrieve all instances of a given HTML element (div, p, li, etc…)
  - #id selector - retrieve the element with id given by id
  - .class selector - retrieve all elements of a given class
- It also defines the usual jQuery pseudo-classes:
  - :first - first element
  - :last - last element
  - :even - even elements (0, 2, 4, …)
  - :odd - odd element (1, 3, 5, …)
  - :eq - a specific element (equals)
  - :lt - less than
  - :gt - greater than

pyQuery

https://pyquery.readthedocs.io/en/latest/

- pyQuery(url=url) or pyQuery(string) to parse a given external url of the html code in a specific string
- .attr(“attr”) returns a specific attribute of a given object.
- .addClass(“bla") - add a css class
- .toggleClass("bla ble”) - toggle class
- .removeClass(“ble") - remove class
- .css(“style”: “value”) - define css style value ("font-size", “15px”)
- .items() - iterate over results

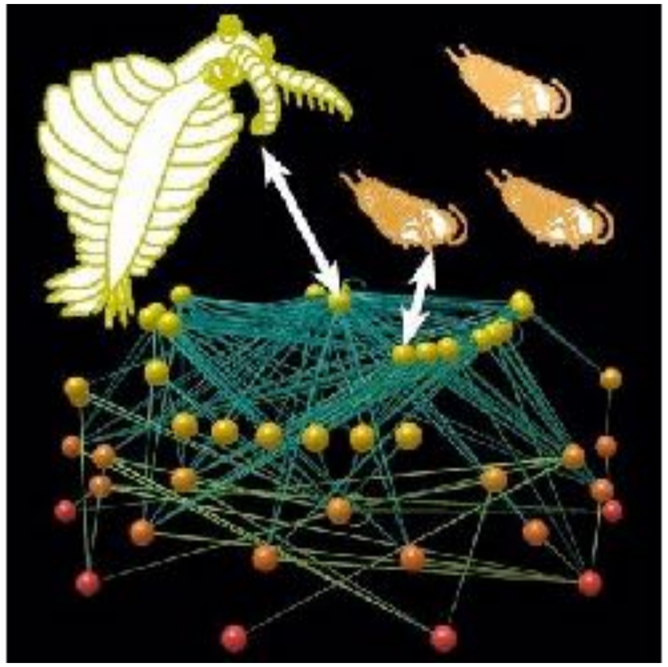
Challenge - pyQuery

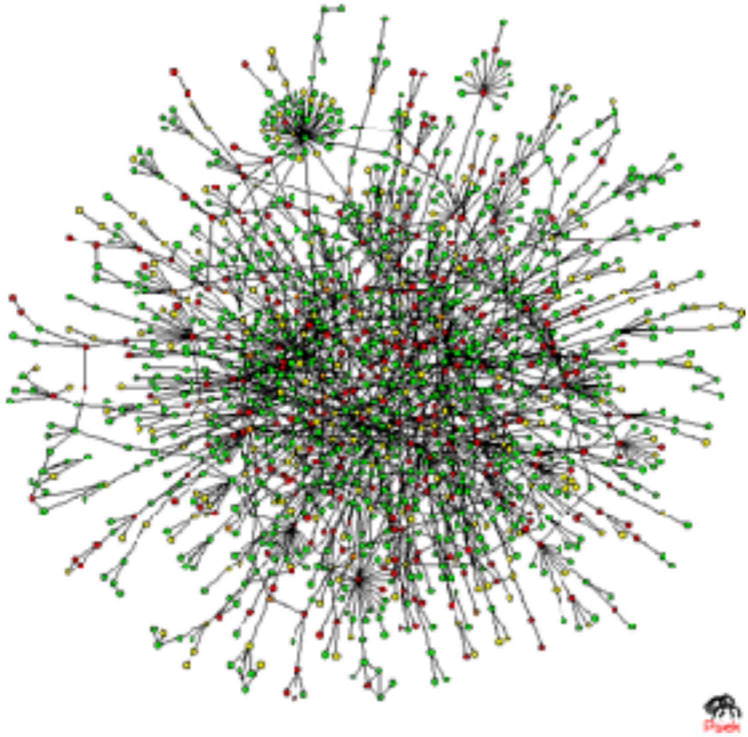
https://pyquery.readthedocs.io/en/latest/

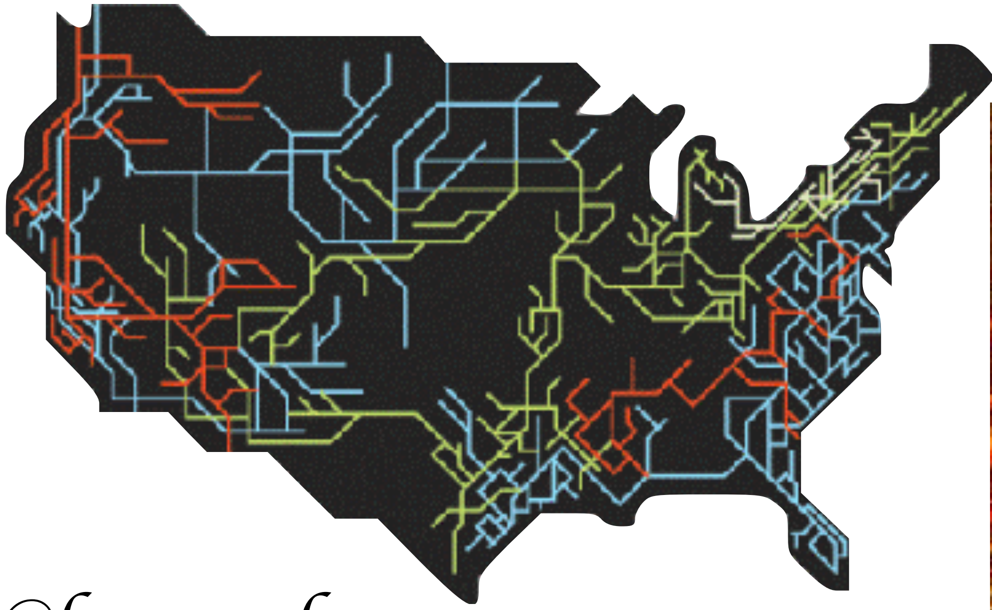
- Extract the title of Feynman's 100 most cited papers from Google Scholar, using pyQuery

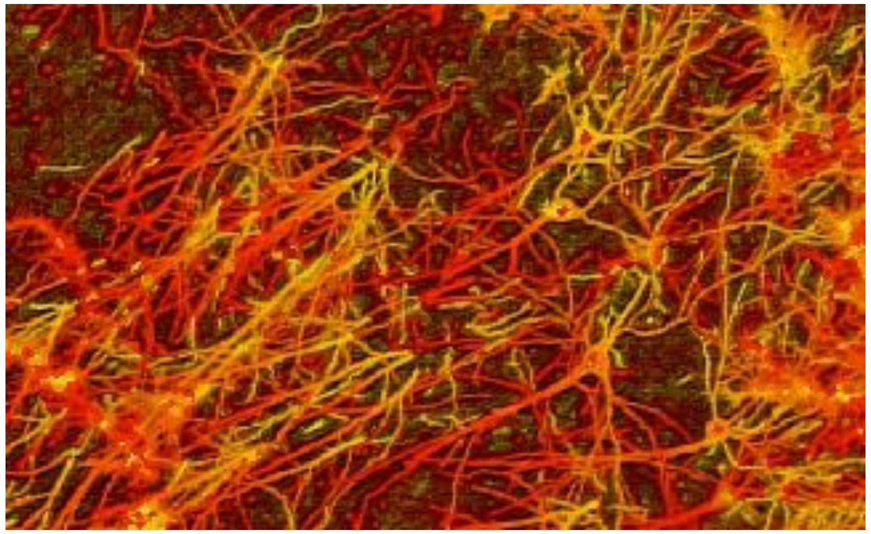
```
from pyquery import PyQuery as pq

url = "http://scholar.google.com/citations?hl=en&user=B7vSqZsAAAAJ&view_op=list_works&pagesize=100"

doc = pq(url=url)
table = doc("table#gsc_a_t")

for row in table("td.gsc_a_t").items():
    print(row("a").text())
```

feynman_pyquery.py



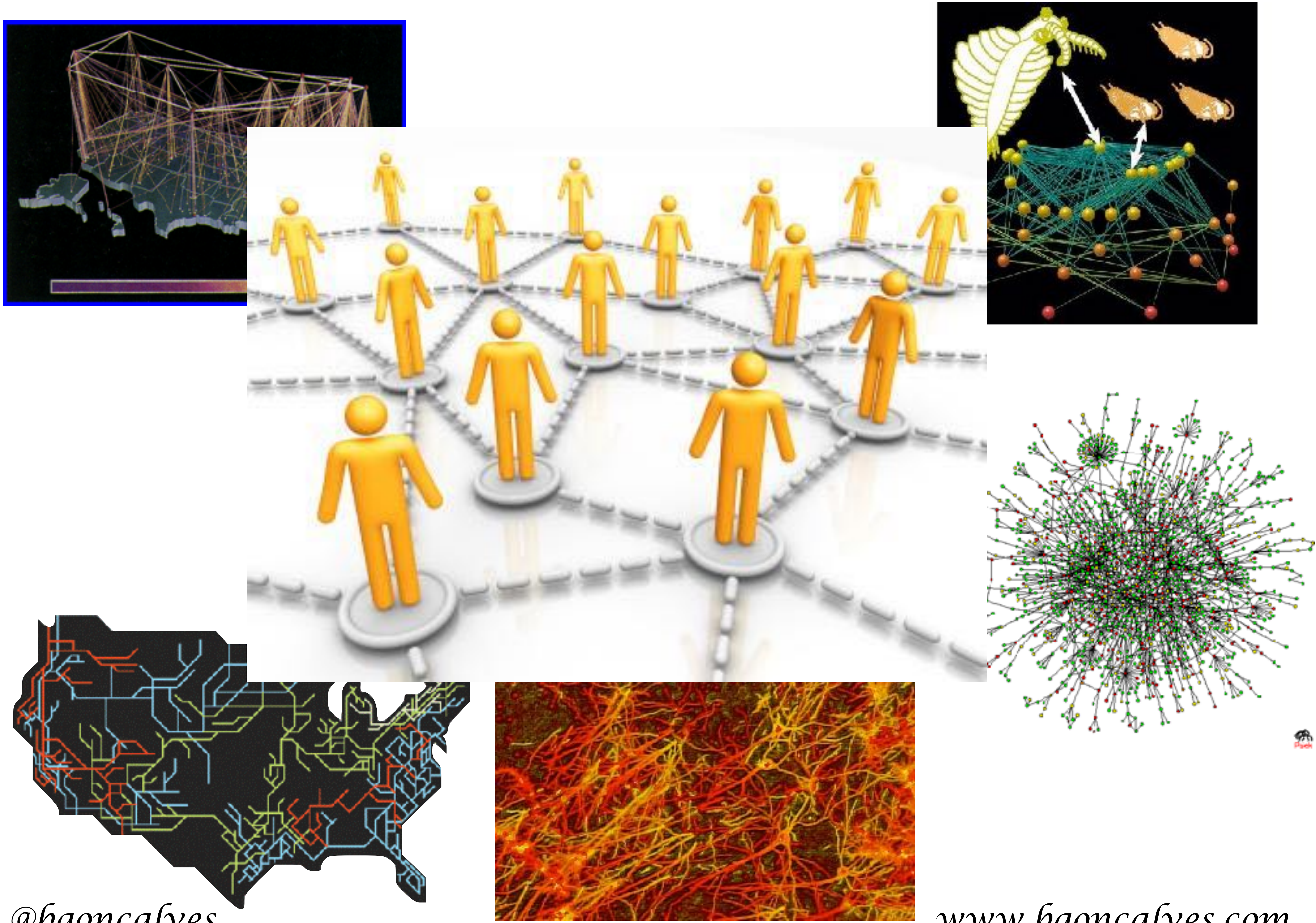




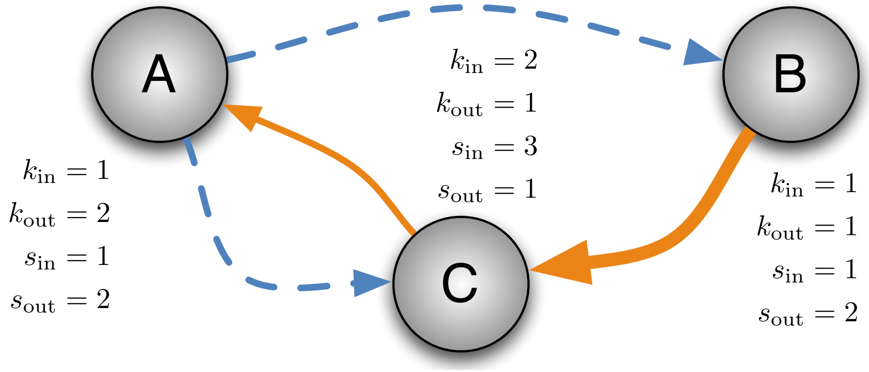


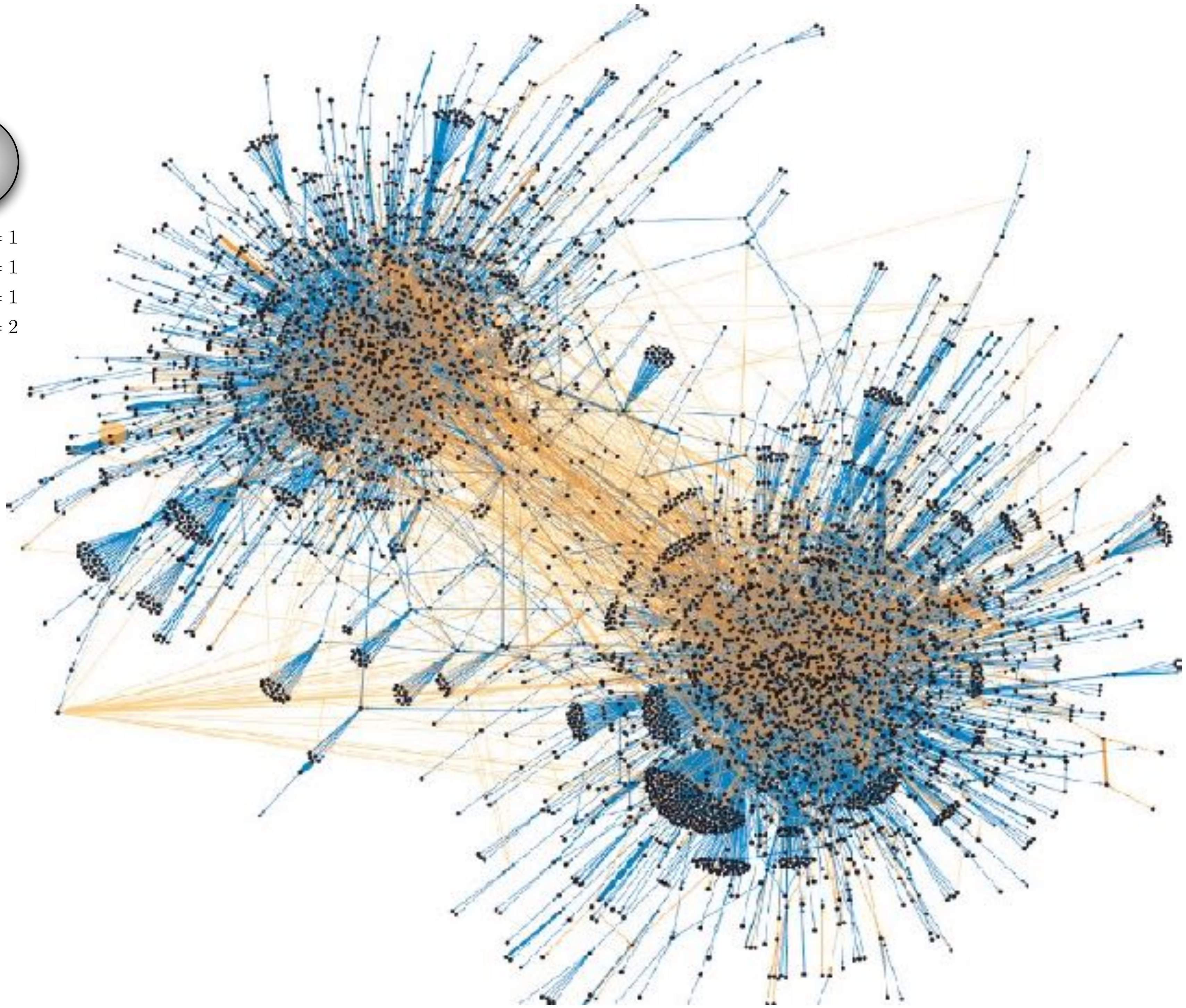
@bgoncalves

www.bgoncalves.com



Information Flow





NetworkX

- High productivity software for complex networks
- Simple Python interface
- Four types of graphs supported:
  - Graph - UnDirected
  - DiGraph - Undirected
  - MultiGraph - Multi-edged Graph
  - MultiDiGraph - Directed Multigraph
- Similar interface for all types of graphs
- Nodes can be any type of Python object - Practical way to manage relationships

Growing Graphs

- .add_node(node_id) Add a single node with ID node_id
- .add_nodes_from() Add a list of node ids
- .add_edge(node_i, node_j) Adds an edge between node_i and node_j
- .add_edges_from() Adds a list of edges. Individual edges are represented by tuples
- .remove_node(node_id)/.remove_nodes_from() Removing a node removes all associated edges
- .remove_edge(node_i, node_j)/.remove_edges_from()

Graph Properties

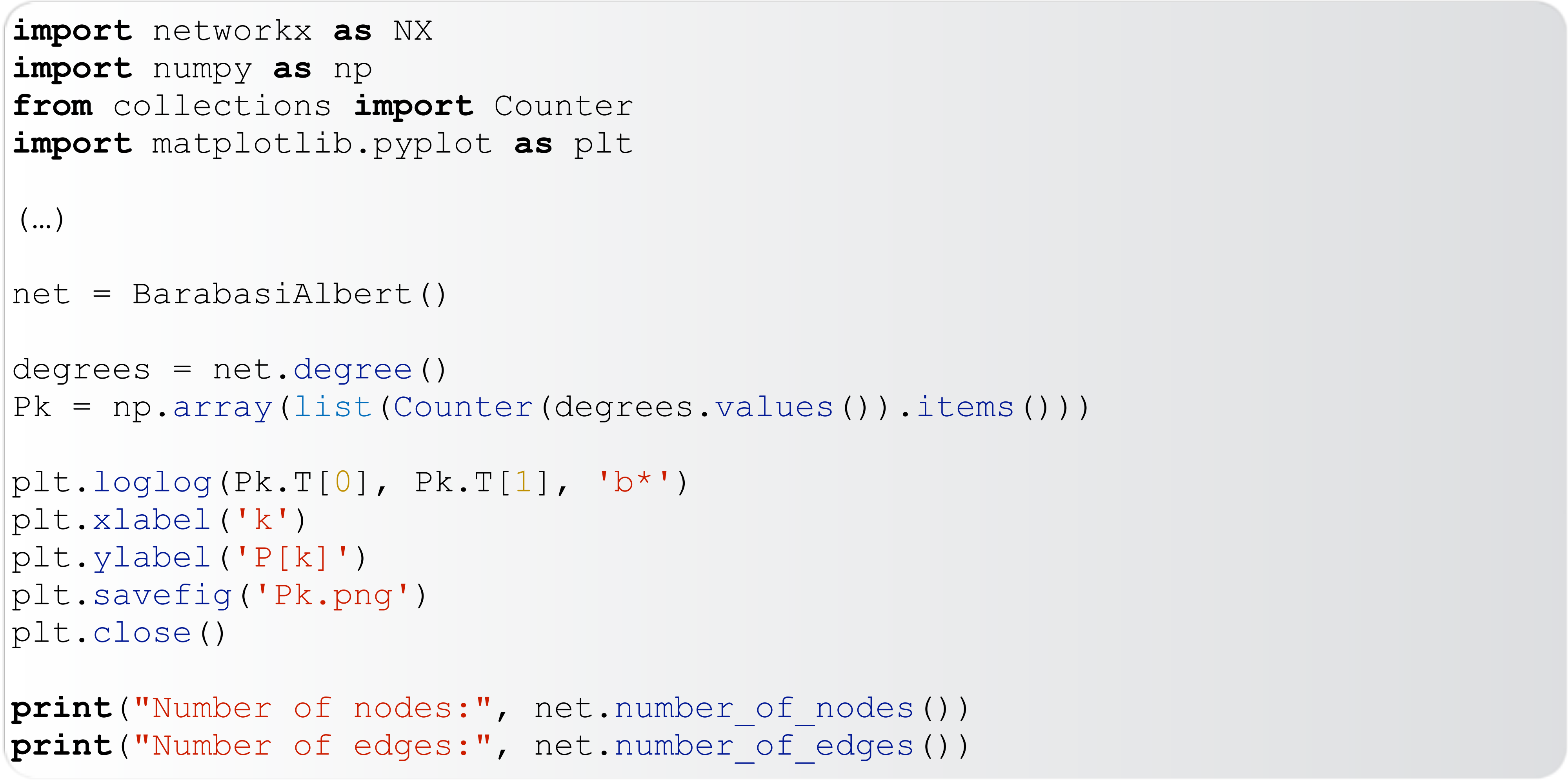
- .nodes() Returns the list of nodes
- .edges() Returns the list of edges
- .degree() Returns a dict with each nodes degree .in_degree()/.out_degree() returns dicts with in/out degree for DiGraphs
- .is_connected() Returns true if the node is connected
- .is_weakly_connected()/.is_strongly_connected() for DiGraph
- .connected_components() A list of nodes for each connected component

NetworkX - Example

```python
import networkx as NX
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt

def BarabasiAlbert(N=1000000):
    G = NX.Graph()

    nodes = range(N)
    G.add_nodes_from(nodes)

    edges = [0,1,1,2,2,0]

    for node_i in range(3, N):
        pos = np.random.randint(len(edges))
        node_j = edges[pos]

        edges.append(node_i)
        edges.append(node_j)

    edges = zip(nodes, edges[1::2])

    G.add_edges_from(edges)

    return G
```

networkx_demo.py

NetworkX - Example



```
import networkx as NX
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt

 (…)

net = BarabasiAlbert()

degrees = net.degree()
Pk = np.array(list(Counter(degrees.values()).items()))

plt.loglog(Pk.T[0], Pk.T[1], 'b*')
plt.xlabel('k')
plt.ylabel('P[k]')
plt.savefig('Pk.png')
plt.close()

print("Number of nodes:", net.number_of_nodes())
print("Number of edges:", net.number_of_edges())
```

networkx_demo.py

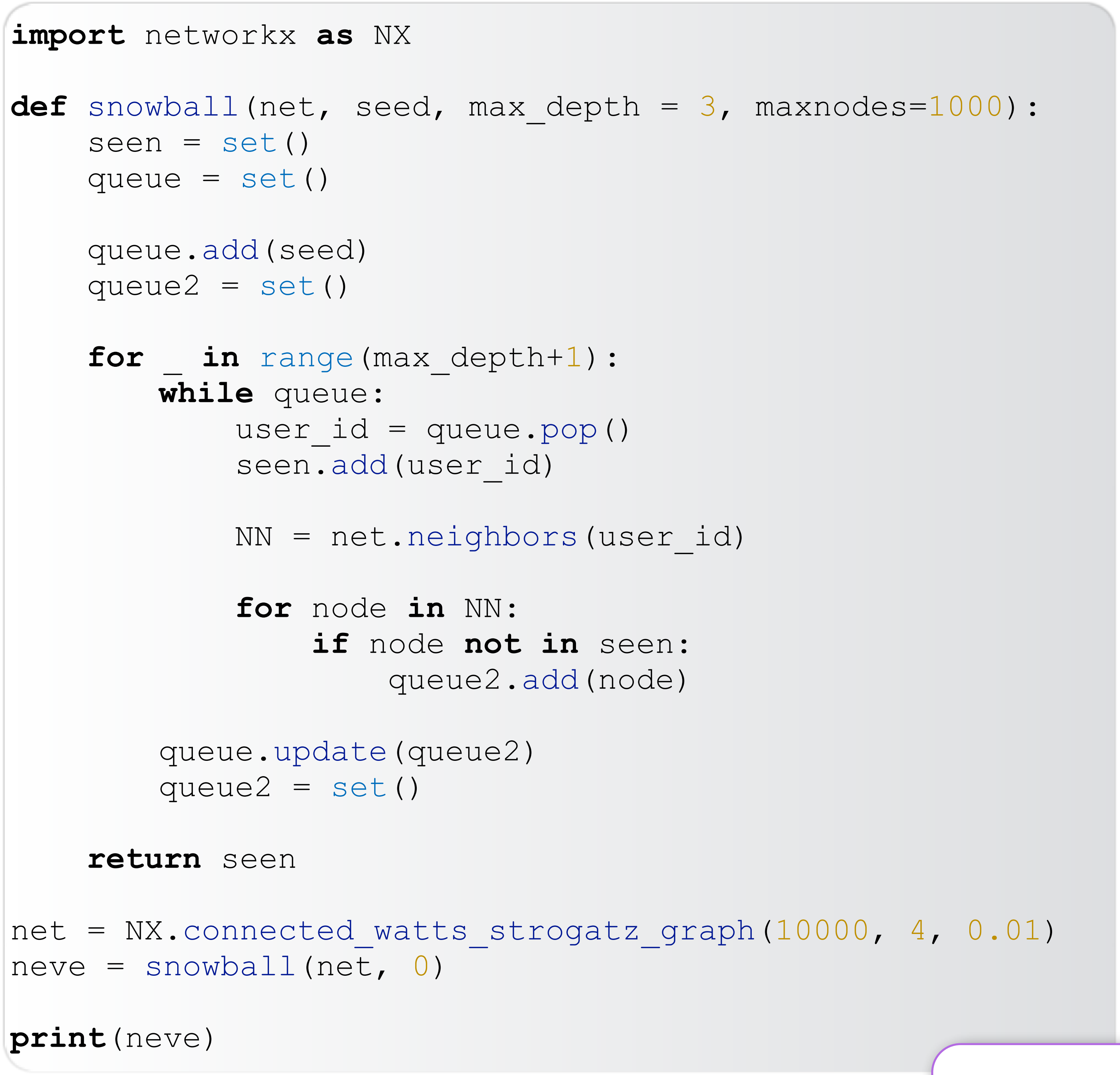
Snowball Sampling

- Commonly used in Social Science and Computer Science

    1. Start with a single node (or small number of nodes)
    2. Get “friends” list
    3. For each friend get the “friend” list
    4. Repeat for a fixed number of layers or until enough users have been connected

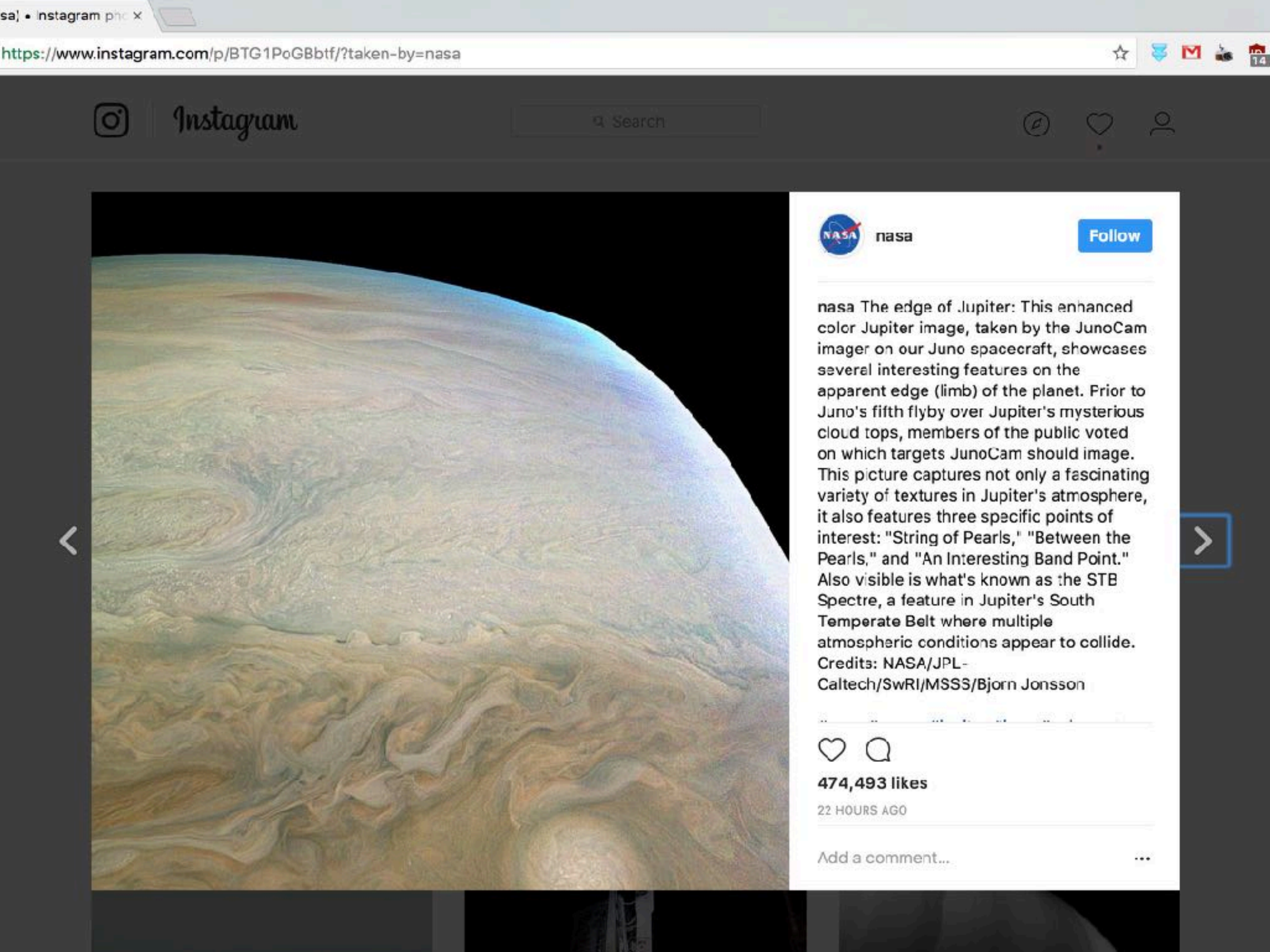
- Generates a connected component from each seed
- Quickly generates a *lot* of data/API calls

Snowball Sampling



```python
import networkx as NX

def snowball(net, seed, max_depth = 3, maxnodes=1000):
    seen = set()
    queue = set()

    queue.add(seed)
    queue2 = set()

    for _ in range(max_depth+1):
        while queue:
            user_id = queue.pop()
            seen.add(user_id)

            NN = net.neighbors(user_id)

            for node in NN:
                if node not in seen:
                    queue2.add(node)

        queue.update(queue2)
        queue2 = set()

    return seen

net = NX.connected_watts_strogatz_graph(10000, 4, 0.01)
neve = snowball(net, 0)

print(neve)
```

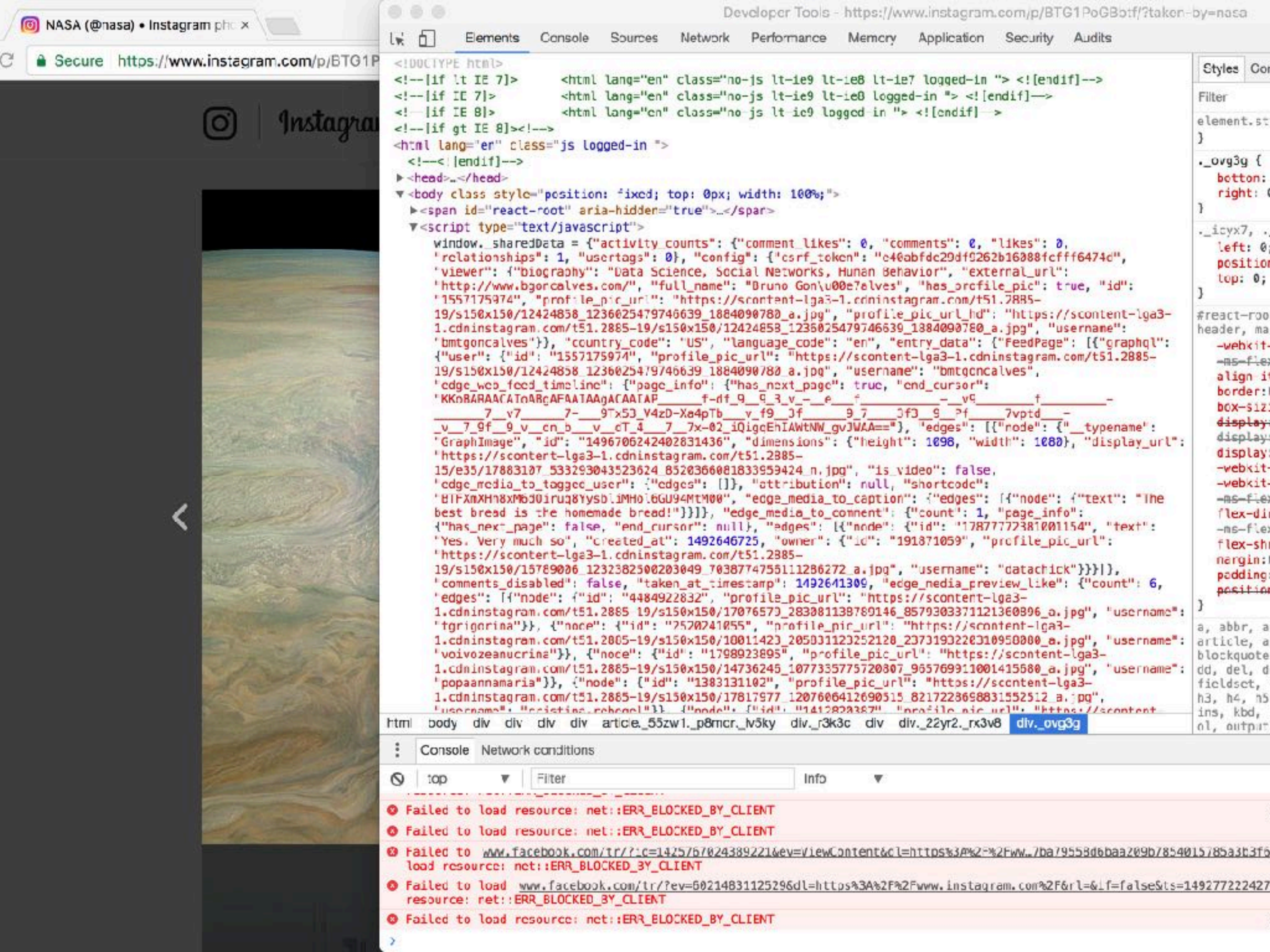
snowball.py



nasa The edge of Jupiter: This enhanced color Jupiter image, taken by the JunoCam imager on our Juno spacecraft, showcases several interesting features on the apparent edge (limb) of the planet. Prior to Juno's fifth flyby over Jupiter's mysterious cloud tops, members of the public voted on which targets JunoCam should image. This picture captures not only a fascinating variety of textures in Jupiter's atmosphere, it also features three specific points of interest: "String of Pearls," "Between the Pearls," and "An Interesting Band Point." Also visible is what's known as the STB Spectre, a feature in Jupiter's South Temperate Belt where multiple atmospheric conditions appear to collide. Credits: NASA/JPL-Caltech/SwRI/MSSS/Bjorn Jonsson

474,493 likes

22 HOURS AGO



NASA (@nasa) • Instagram phc ×
Secure https://www.instagram.com/p/BTG1P
Instagra
Developer Tools - https://www.instagram.com/p/BTG1PoGBotf/?taken-by=nasa
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
Styles
Filter
html body div div div div article._55zw1._p8mcr._lv5ky div._r3k3c div div._22yr2._rx3v8 div._ovg3g
Console
Network conditions
top
Filter
Info

```
<!DOCTYPE html>
<!--[if lt IE 7]>      <html lang="en" class="no-js lt-ie9 lt-ie8 lt-ie7 logged-in "> <![endif]-->
<!--[if IE 7]>         <html lang="en" class="no-js lt-ie9 lt-ie8 logged-in "> <![endif]-->
<!--[if IE 8]>         <html lang="en" class="no-js lt-ie9 logged-in "> <![endif]-->
<!--[if gt IE 8]><!-->
<html lang="en" class="js logged-in ">
  <!--<![endif]-->
▶<head>…</head>
▼<body class style="position: fixed; top: 0px; width: 100%;">
  ▶<span id="react-root" aria-hidden="true">…</span>
  ▼<script type="text/javascript">
```

```
window._sharedData = {"activity_counts": {"comment_likes": 0, "comments": 0, "likes": 0,
"relationships": 1, "usertags": 0}, "config": {"csrf_token": "c40abfdc29df9262b16088fcfff6474d",
"viewer": {"biography": "Data Science, Social Networks, Human Behavior", "external_url":
"http://www.bgoncalves.com/", "full_name": "Bruno Gon\u00e7alves", "has_profile_pic": true, "id":
"1557175974", "profile_pic_url": "https://scontent-lga3-1.cdninstagram.com/t51.2885-
19/s150x150/12424858_1236025479746639_1884090780_a.jpg", "profile_pic_url_hd": "https://scontent-lga3-
1.cdninstagram.com/t51.2885-19/s150x150/12424858_1236025479746639_1884090780_a.jpg", "username":
"bmtgoncalves"}}, "country_code": "US", "language_code": "en", "entry_data": {"FeedPage": [{"graphql":
{"user": {"id": "1557175974", "profile_pic_url": "https://scontent-lga3-1.cdninstagram.com/t51.2885-
19/s150x150/12424858_1236025479746639_1884090780_a.jpg", "username": "bmtgoncalves",
"edge_web_feed_timeline": {"page_info": {"has_next_page": true, "end_cursor":
"KKoBABAACAToABgAEAAIAAgACAAIAP______f-df_9__9_3_v_-__e___f___________-__v9________f_________-
_______7__v7______7-___9Tx5J_V4zD-Xa4pTb___v_f9__Jf______9_7____Jf3__9__?f_____7vptd___-
_v__7_9f__9_v__cn_b___v__cT_4___7__7x-02_iQigcEhIAWtNW_gvJWAA=="}, "edges": [{"node": {"__typename":
"GraphImage", "id": "1496706242402831436", "dimensions": {"height": 1098, "width": 1080}, "display_url":
"https://scontent-lga3-1.cdninstagram.com/t51.2885-
15/e35/17883107_533293043523624_8520366081833959424_n.jpg", "is_video": false,
"edge_media_to_tagged_user": {"edges": []}, "attribution": null, "shortcode":
"BTFXmXHn8xM6d0iruq8YysbliMHol6GU94MtM00", "edge_media_to_caption": {"edges": [{"node": {"text": "The
best bread is the homemade bread!"}}]}, "edge_media_to_comment": {"count": 1, "page_info":
{"has_next_page": false, "end_cursor": null}, "edges": [{"node": {"id": "17877772381001154", "text":
"Yes. Very much so", "created_at": 1492646725, "owner": {"id": "191871059", "profile_pic_url":
"https://scontent-lga3-1.cdninstagram.com/t51.2885-
19/s150x150/15789006_1232382500203049_7038774755111286272_a.jpg", "username": "datachick"}}}]},
"comments_disabled": false, "taken_at_timestamp": 1492641309, "edge_media_preview_like": {"count": 6,
"edges": [{"node": {"id": "4484922832", "profile_pic_url": "https://scontent-lga3-
1.cdninstagram.com/t51.2885-19/s150x150/17076570_283081138789146_8579303371121360896_a.jpg", "username":
"tgrigorina"}}, {"node": {"id": "2520241055", "profile_pic_url": "https://scontent-lga3-
1.cdninstagram.com/t51.2885-19/s150x150/10011423_2050311232521280_2373193228310958080_a.jpg", "username":
"voivozeanucrina"}}, {"node": {"id": "1798923895", "profile_pic_url": "https://scontent-lga3-
1.cdninstagram.com/t51.2885-19/s150x150/14736245_1077335775720807_9655769911001415680_a.jpg", "username":
"popaannamaria"}}, {"node": {"id": "1383131102", "profile_pic_url": "https://scontent-lga3-
1.cdninstagram.com/t51.2885-19/s150x150/17817977_1207606412690515_8217228698831552512_a.jpg",
"username": "cristina.rebegel"}}, {"node": {"id": "1412823387", "profile_pic_url": "https://scontent-
```

```
element.st
}
._ovg3g {
  bottom:
  right:
}
._icyx7, .
  left: 0
  position
  top: 0;
}
#react-roo
header, ma
  -webkit-
  -ms-flex
  align-it
  border:
  box-sizi
  display
  display
  display:
  -webkit-
  -webkit-
  -ms-flex
  flex-dir
  -ms-flex
  flex-shr
  margin:
  padding:
  position
}
a, abbr, a
article, a
blockquote
dd, del, d
fieldset,
h3, h4, h5
ins, kbd,
ol, output
```

```
⊗ Failed to load resource: net::ERR_BLOCKED_BY_CLIENT
⊗ Failed to load resource: net::ERR_BLOCKED_BY_CLIENT
⊗ Failed to  www.facebook.com/tr/?id=1425767024389221&ev=ViewContent&dl=https%3A%2F%2Fww…7ba79558d6baa209b78540157 85a3b3f6
  load resource: net::ERR_BLOCKED_BY_CLIENT
⊗ Failed to load  www.facebook.com/tr/?ev=6021483112529&dl=https%3A%2F%2Fwww.instagram.com%2F&rl=&if=false&ts=149277222427
  resource: net::ERR_BLOCKED_BY_CLIENT
⊗ Failed to load resource: net::ERR_BLOCKED_BY_CLIENT
>
```

Instagram

```
import requests
from bs4 import BeautifulSoup

url = "https://www.instagram.com/p/BTG1PoGBbtf/"

jsfuncs = {
    "true": True,
    "false": False,
    "null": None
}

magic_string = "window._sharedData = "
offset = len(magic_string)

page = requests.get(url).content
soup = BeautifulSoup(page, "lxml")

scripts = soup.findAll("script")

for script in scripts:
    if script.text.startswith(magic_string):
        text = script.text[offset:-1]
        data = eval(text, jsfuncs)
        break

comments = data["entry_data"]["PostPage"][0]["graphql"]["shortcode_media"]["edge_media_to_comment"]["edges"]

for comment in comments:
    print(comment["node"]["owner"]["username"], "==>", comment["node"]["text"].encode('utf8','replace').decode())
```

instagram.py