This notebook uses `scholarly` API (https://github.com/scholarly-python-package/scholarly (https://github.com/scholarly-python-package/scholarly)) to enrich CS faculty info

In [3]:
```
from scrap_cs_faculty import *
```

```
import re
def normalize_str(text, non_alpha_numeric=r'[^a-z0-9]', sep="_"):
    # Replace all non-alphanumeric chars with underscores
    words = re.sub(non_alpha_numeric, sep, text.lower()).split(sep)
    return sep.join([w for w in words if w])
```

```
text = "!This has     many spaces!"
normalized = normalize_str(text)
print(normalized)
```

```
def get_scholar_page(scholar_id, base_url=SCHOOL_DICT["Google-Scholar"]["url"], lang=LANG):
    if not scholar_id:
        return ""
    return f"{base_url}/citations?user={scholar_id}&hl={lang}&oi=ao"
```

In [14]:
```
org_list = ["CMU-CS","Cornell-CS","MIT-AID","MIT-CS","Stanford-CS","UCB-CS","UIUC-CS",]
```

In [17]:
```
org = org_list[4]  # stanford
```

In [18]:
```
org2 = org.split("-")[0]
```

In [19]:
```
file_xlsx = f"faculty-{org}.xlsx"
xlsxf = pd.ExcelFile(file_xlsx)
```

In [20]:
```
xlsxf.sheet_names
```

Out[20]:
```
['Faculty']
```

In [21]:
```
df = xlsxf.parse('Faculty')
```

In [22]:
```
df
```

		Faculty						
142	Hamed Nemati	Visiting and Acting Faculty	NaN	NaN	NaN	NaN	NaN	https://hnemati.gitl
143	Dolière Francis Somé	Visiting and Acting Faculty	NaN	NaN	NaN	NaN	NaN	
144	Marco Vassena	Visiting and Acting Faculty	NaN	NaN	NaN	NaN	NaN	https://webspace.science.uu.nl/mva
145	Zhikun Zhang	Visiting and Acting Faculty	NaN	NaN	NaN	NaN	NaN	http://zhangzh

In [23]:
```
names = df["name"].to_list()
```

In [133]:
```
len(names), names
```

Out[133]:
```
(146,
 ['Sara Achour',
  'Maneesh Agrawala',
  'Alex Aiken',
  'Nima Anari',
  'Clark Barrett',
  'Gill Bejerano',
  'Michael Bernstein',
  'Jeannette Bohg',
  'Dan Boneh',
  'Adam Bouland',
  'Emma Brunskill',
  'Moses Charikar',
  'Ron Dror',
  'Zakir Durumeric',
  'Dawson Engler',
  'Stefano Ermon',
  'Kayvon Fatahalian',
  'Ron Fedkiw',
```

1 fetch google scholar data for all CS faculty

In [119]:
```
SCHOLAR_HEADER = [
    'name',
    'affiliation',
    'interests',
    'num_papers',
    'num_coauthors',
    'citedby',
    'hindex',
    'i10index',
    'citedby5y',
    'hindex5y',
    'i10index5y',
    'scholar_id',
    'url_author',
    'url_picture',
    'url_homepage',
    'file_author']
```

In [120]:
```
ntest = -1  #  2 #
```

In [127]:

```
pub_data = []
for n, name in enumerate(names[:ntest]):
    if n < 5: continue
    print(f"n, name = {n}, {name} ...")

    author_org = f"{name} {org2}"
    norm_auth_org = normalize_str(author_org)
    file_author = f"data/GScholar_{norm_auth_org}.json"

    try:
        search_query = scholarly.search_author(author_org)
        init_result = next(search_query)
    except Exception as ex:
        print(str(ex))
        init_result = None

    if init_result is None:
        print(f"Failed search_author()")
        continue

    scholar_id = init_result.get("scholar_id", "")
    if not scholar_id:
        print(f"Missing scholar_id")
        continue

    # fetch data
    author = scholarly.fill(init_result)

    author_dict = {}
    # fill data cell
    author_dict["name"] = name
    author_dict["file_author"] = file_author
    author_dict["scholar_id"] = scholar_id
    author_dict["affiliation"] = author.get("affiliation", "")
    author_dict["interests"] = "; ".join(author.get("interests", []))
    author_dict["url_author"] = get_scholar_page(scholar_id)
    author_dict["url_picture"] = author.get("url_picture", "")
    author_dict["url_homepage"] = author.get("homepage", "")
    author_dict["citedby"] = author.get("citedby", 0)
    author_dict["citedby5y"] = author.get("citedby5y", 0)
    author_dict["hindex"] = author.get("hindex", 0)
    author_dict["hindex5y"] = author.get("hindex5y", 0)
    author_dict["i10index"] = author.get("i10index", 0)
```

```
    author_dict["i10index5y"] = author.get("i10index5y", 0)
    author_dict["num_papers"] = len(author.get("publications", []))
    author_dict["num_coauthors"] = len(author.get("coauthors", []))

    # fill row
    author_data = []
    for c in SCHOLAR_HEADER:
        author_data.append(author_dict.get(c))

    # accumulate row
    pub_data.append(author_data)

    # persist author data
    with open(Path(file_author), "w", encoding="utf-8") as f:
        f.write(json.dumps(author))

    delay = randint(1,5)
    sleep(delay)
```

```
Failed search_author()
n, name = 137, Silvio Savarese ...
n, name = 138, Sebastian Thrun ...

Failed search_author()
n, name = 139, Shahar Dobzinski ...

Failed search_author()
n, name = 140, Aurore Fass ...
n, name = 141, Chris Hahn ...

Failed search_author()
n, name = 142, Hamed Nemati ...
n, name = 143, Dolière Francis Somé ...

Failed search_author()
n, name = 144, Marco Vassena ...

Failed search_author()
```

```
StopIteration: n = 5 when Google scholar profile not found
```

```
author_dict
```

```
{'name': 'Sara Achour',
 'file_author': 'GScholar_sara_achour_stanford.json',
 'scholar_id': 'J-5mJ7AAAAAJ',
 'affiliation': 'Stanford University',
 'interests': 'programming languages; compilers; emerging hardware technologies',
 'url_author': 'https://scholar.google.com/citations?user=J-5mJ7AAAAAJ&hl=en&oi=ao',
 'url_picture': 'https://scholar.google.com/citations?view_op=medium_photo&user=J-5mJ7AAAAAJ',
 'homepage': 'https://www.sara-achour.me/',
 'citedby': 769,
 'citedby5y': 541,
 'hindex': 5,
 'hindex5y': 5,
 'i10index': 5,
 'i10index5y': 4,
 'num_coauthors': 5,
 'num_papers': 12}
```

```
author
```

In [128]:
```
len(pub_data)
```

Out[128]: 102

2 write out xlsx

In [129]:
```
df_out = pd.DataFrame(pub_data, columns=SCHOLAR_HEADER)
```

In [132]:
```
df_out.head()
```

Out[132]:

	name	affiliation	interests	num_papers	num_coauthors	citedby	hindex	i10index	citedby5y	hindex5y	i10index5y	s
0	Michael Bernstein	Associate Professor, Stanford University	Human-computer interaction; social computing	245	73	63473	61	133	51505	53	118	zkhI
1	Jeannette Bohg	Assistant Professor, Stanford University	Robotics; Multi-Modal Perception; Machine Lear...	135	96	7216	36	76	6149	33	67	rjnJn
2	Dan Boneh	Professor of Computer Science, Stanford Univer...	Cryptography; Computer Security; Computer Scie...	475	18	101996	132	302	38220	89	249	MwLqC
3	Adam Bouland	Stanford University	Quantum Computing; Theoretical Computer Science	33	44	889	15	18	762	14	17	61uf9
4	Emma Brunskill	Associate Professor of Computer Science, Stanf...	Reinforcement Learning; Machine Learning; Deci...	220	19	10839	49	120	8636	43	105	HaN8b

In [131]:
```
# import xlsxwriter
file_xlsx = f"data/cs-faculty-gscholar-{org2}-{n}.xlsx"
writer = pd.ExcelWriter(Path(file_xlsx), engine='xlsxwriter')
df_out.to_excel(writer, sheet_name=org2, index=False)
writer.save()
```

```
C:\Users\p2p2l\AppData\Local\Temp\ipykernel_15664\4019838328.py:5: FutureWarning: save is not part of the pu
blic API, usage can give unexpected results and will be removed in a future version
  writer.save()
```

In [ ]:

In [ ]: