Review

Building ETL pipelines with Python and SQL

Context:

- E xtract data from various sources
- T ransform data into compatible format for loading
- L oad data into database

Skills we've learned:

- Python
- SQL
- API
- Vectorized operations (Pandas)

Functions

define a function

```
def hello():
    print("Hello world")

hello()
```

with argument

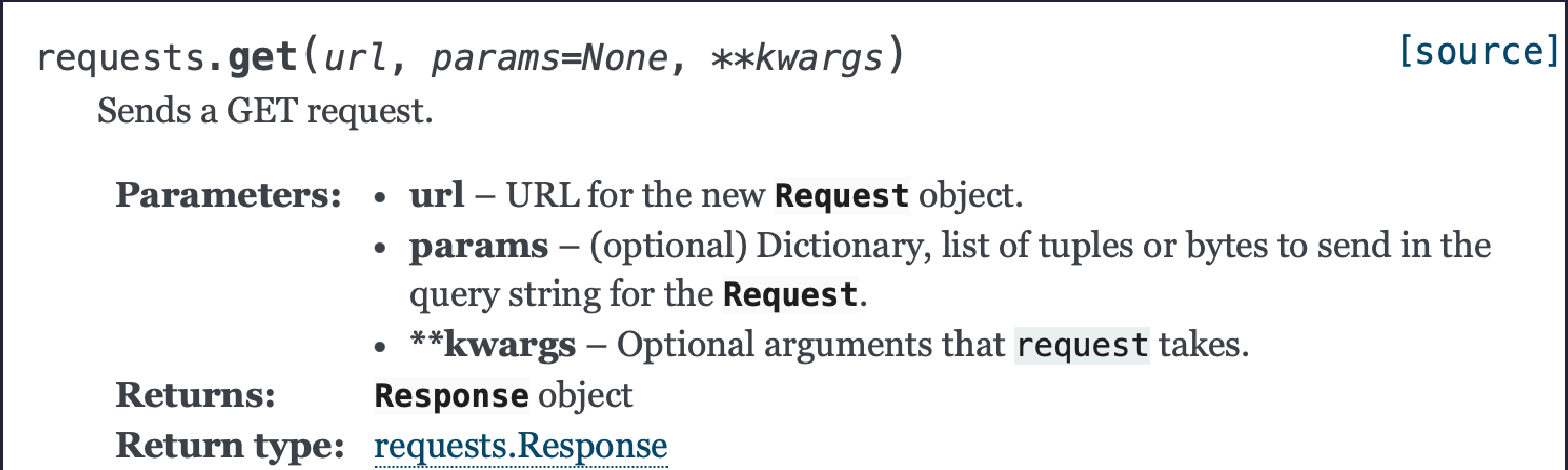
```python
def hello(to):
    print(f"Hello {to}")

hello("harry")              # Hello harry
hello(to="harry")           # Hello harry
hello()                     # error
```

argument with default value

```python
def hello(to="world"):
    print(f"Hello {to}")

hello("harry")              # Hello harry
hello(to="harry")           # Hello harry
hello()                     # Hello world
```

Positional and keyword arguments

```python
def hello(to_1st, to_2nd):
    print(f"Hello {to_1st} and {to_2nd}")

hello("harry", "hermione")
hello("hermione", "harry")
hello(to_2nd="hermione", to_1st="harry")
```



`requests.get(url, params=None, **kwargs)` [source]

Sends a GET request.

Parameters:

- **url** – URL for the new `Request` object.
- **params** – (optional) Dictionary, list of tuples or bytes to send in the query string for the `Request`.
- ****kwargs** – Optional arguments that `request` takes.

Returns: `Response` object

Return type: requests.Response

```python
url = "https://www.google.com/search"
params = {"q": "harry potter"}
headers = {"User-Agent": "Mozilla/5.0"}
# 1
requests.get(url, params)
# 2
requests.get(params, url)
# 3
requests.get(params=params, url=url)
# 4
requests.get(url, params, headers)
# 5
requests.get(url, headers=headers, params=params)
```

Collection arguments

```python
def hello(to: list):
    to_1st = to[0]
    to_2nd = to[1]
    print(f"Hello {to_1st} and {to_2nd}")

hello(["harry", "hermione"])
hello(to=["harry", "hermione"])
```

```python
def hello(to: dict):
    to_1st = to['1st']
    to_2nd = to['2nd']
    print(f"Hello {to_1st} and {to_2nd}")

hello({"1st": "harry", "2nd": "hermione"})
hello(to={"1st": "harry", "2nd": "hermione"})
```

```python
def hello(to):
    to_1st_name = to['1st']['name']
    to_1st_house = to['1st']['house']
    to_2nd_name = to['2nd']['name']
    to_2nd_house = to['2nd']['house']
    message = f"""
        Hello {to_1st_name} from {to_1st_house} and
        {to_2nd_name} from {to_2nd_house}
    """
    print(message)

students = {
    "1st": {"name": "harry", "house": "gryffindor"},
    "2nd": {"name": "hermione", "house": "gryffindor"}
}
hello(students)
```

Variables and data types

```
a = 1

b = "hello"

c = [1, 2, 3]

d = (1, 2, 3)

e = {"a": 1, "b": 2}

f = {"a": [1, 2, 3], "b": {"c": 4, "d": 5}}

g = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
```

Methods for each data type

```
b = "hello"
b.upper().lower().capitalize().title().strip()

c = [1, 2, 3]
c.append(4)
c.sort()

e = {"a": 1, "b": 2}
e.keys()
e.values()
e.items()
e.keys().values().items()        # error

g = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
g.head()
g.agg({"a": ["mean", "std"], "b": ["min", "max"]})
g.merge(g, on="a").dropna()

conn = sqlite3.connect('harrypoter.db')
conn.execute("SELECT * FROM students").fetchall()
```

Control - conditional

```python
if a > 0:
    print("a is positive")
elif a < 0:
    print("a is negative")
else:
    print("a is zero")
```

Boolean expressions: True or False

```
print(2 > 1)

print(5 % 2 == 0)

print(not 5 % 2 == 0)

print(2 > 1 or 2 < 1)

print(True or False)

print(5 % 2 == 0 or 5 % 3 == 0)

print(2 > 1 and 2 < 1)

print(True and False)

print(False and False)
```

Conditions = boolean expressions

```python
if x % 2 == 0: # True or False
    print("x is even")

if x > y: # True or False
    print("x is greater than y")

if score >= 85: # True or False
    print("A")

if True:
    print("always get printed")

if False:
    print("never get printed")
```

Control - for loop

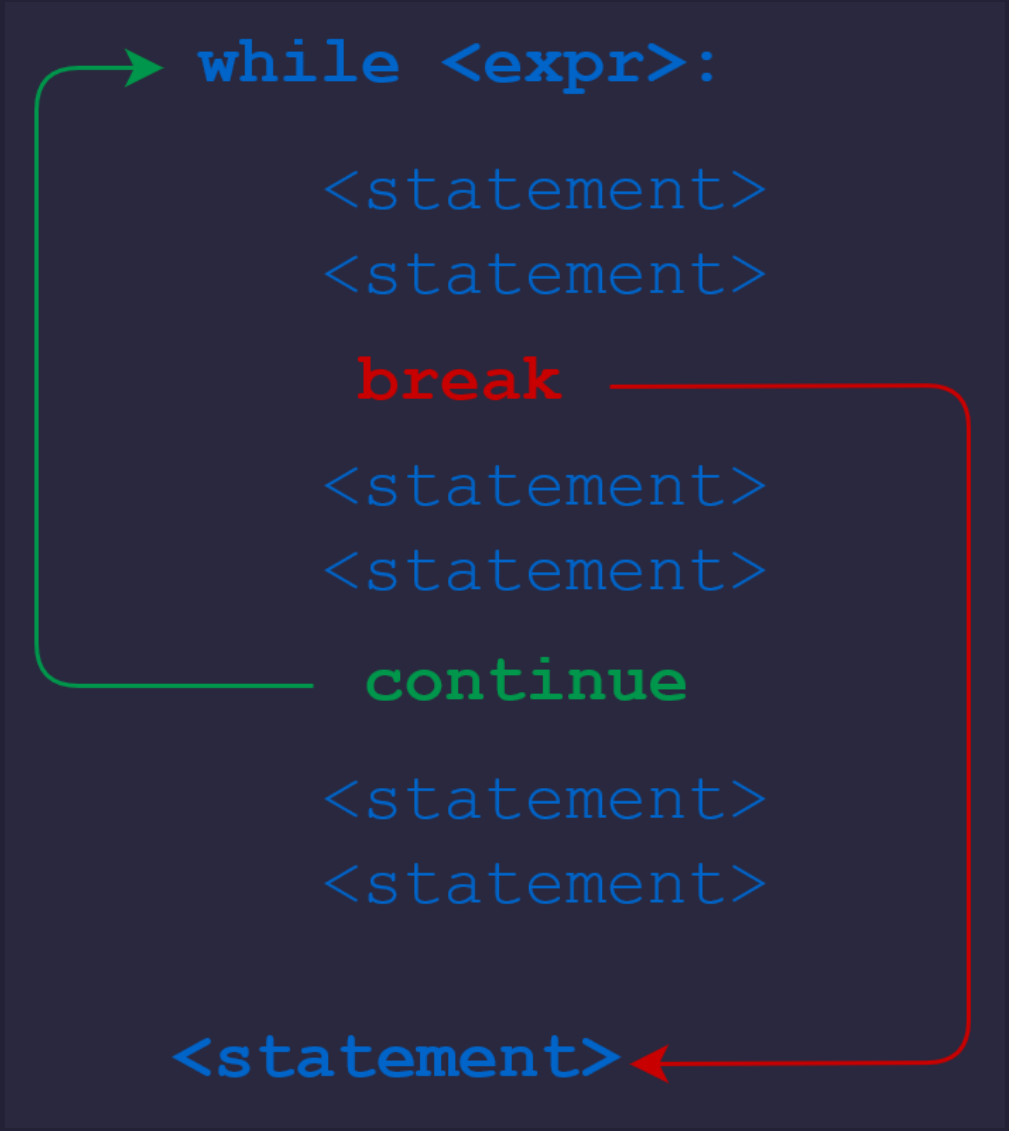
```python
for i in range(10): # 0, 1, 2, ..., 9
    print(i)

for row in data: # data is a list
    print(row)

for key, value in data.items(): # data is a dictionary
    print(key, value)
```

Control - while loop

```
# initialize a counter
i = 0
while i < 10:    # condition
    print(i)
    i += 1       # update the counter
```

```
# infinite loop
while True:
    print("hello")
```

Control - break and continue



```
while <expr>:
    <statement>
    <statement>
    break
    <statement>
    <statement>
    continue
    <statement>
    <statement>

<statement>
```

Control - break and continue

```python
for i in range(10):
    if i == 5:
        break
    print(i)
```

```python
for i in range(10):
    if i == 5:
        continue
    print(i)
```

```python
def count_to_five():
    for i in range(10):
        if i == 5:
            return
        print(i)
count_to_five()
```

Handling data (SQL vs. Pandas)

- Filtering
- Sorting
- Aggregation
- Grouping
- Joining

Filtering

SQL

```sql
SELECT * FROM students WHERE age = 10;
SELECT first_name, house FROM students WHERE age > 10;
SELECT * FROM students WHERE age in (10, 11);
```

Pandas - `query`

```python
df.query('age == 10')
df.query('age > 10')[['first_name', 'house']]
df.query('age in (10, 11)')
```

Pandas - `loc`

```python
df.loc[df['age'] == 10]
df.loc[df['age'] > 10, ['first_name', 'house']]
df.loc[df['age'].isin([10, 11])]
```

Sorting

SQL

```sql
SELECT * FROM students ORDER BY age;
SELECT * FROM students ORDER BY age desc;
SELECT * FROM students ORDER BY age, first_name;
```

Pandas

```python
df.sort_values(by='age')
df.sort_values(by='age', ascending=False)
df.sort_values(by=['age', 'first_name'])
```

Aggregation

single aggregation function

```
SELECT AVG(age) FROM students;
```

```
df['age'].mean()
df['age'].agg('mean')
df.agg({'age': 'mean'})
```

multiple aggregation functions

```
SELECT AVG(age), MAX(age) FROM students;
```

```
df.agg({'age':['mean', 'max']})
```

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.aggregate.html

Grouping

SQL

```
SELECT house_id, AVG(age) FROM students GROUP BY house_id;
SELECT house_id, AVG(age), MAX(age) FROM students GROUP BY house_id;
```

Pandas

```
df.groupby('house_id').agg({'age': 'mean'})
df.groupby('house_id').agg({'age': ['mean', 'max']})
```

https://realpython.com/pandas-groupby/

Joining

SQL

```
SELECT * FROM posts JOIN stocks ON posts.ticker = stocks.ticker;
```

Pandas

```
pd.merge(posts, stocks, left_on='ticker', right_on='ticker', how='inner')
pd.merge(posts, stocks, left_on='ticker', right_on='ticker')
pd.merge(posts, stocks, on='ticker')
posts.merge(stocks, on='ticker')
```

Connecting to a database in Python

```
import sqlite3

conn = sqlite3.connect('harrypoter.db')
```

Execute a query (DQL)

SQL

```sql
SELECT * FROM students;
```

Python

```python
conn.execute("SELECT * FROM students").fetchone()
conn.execute("SELECT * FROM students").fetchmany(5)
conn.execute("SELECT * FROM students").fetchall()
```

Execute a query (DDL)

```
query = """
    CREATE TABLE students (
        id INTEGER PRIMARY KEY,
        name TEXT,
        house TEXT,
        age INTEGER
    )
"""
conn.execute(query)
```

Execute a query (DML)

```
query = """
    INSERT INTO students (id, name, house, age)
    VALUES (1, 'Harry Potter', 'Gryffindor', 11)
"""
conn.execute(query)
```

Execute a query (DML) dynamically

Option 1 - tuple

```
data = (2, 'Hermione Granger', 'Gryffindor', 11)
query = f"INSERT INTO students VALUES {data}"
conn.execute(query)
```

Option 2 - tuple with params

```
data = (2, 'Hermione Granger', 'Gryffindor', 11)
query = "INSERT INTO students VALUES (?, ?, ?, ?)"
conn.execute(query, data)
```

Option 3 - dictionary with params

```
data = {"id": 2, "name": 'Hermione Granger', "house": 'Gryffindor', "age": 11}
query = "INSERT INTO students VALUES (:id, :name, :house, :age)"
conn.execute(query, data)
```

REST API syntax

`https://itunes.apple.com/search?entity=movie&term=avengers&limit=1`

- endpoint: `itunes.apple.com/`
- path: `search`
- query parameters: `?entity=movie&term=avengers&limit=1`

https://www.ibm.com/docs/en/informix-servers/12.10?topic=api-rest-syntax

Extract data from APIs using `requests`

`https://itunes.apple.com/search?entity=movie&term=avengers&limit=1`

```python
import requests

url = "https://itunes.apple.com/search"

params = {
    "entity": "movie",
    "term": "avengers",
    "limit": 1
}

response = requests.get(url, params=params)

print(response.json())
```

API response in JSON (JavaScript Object Notation)

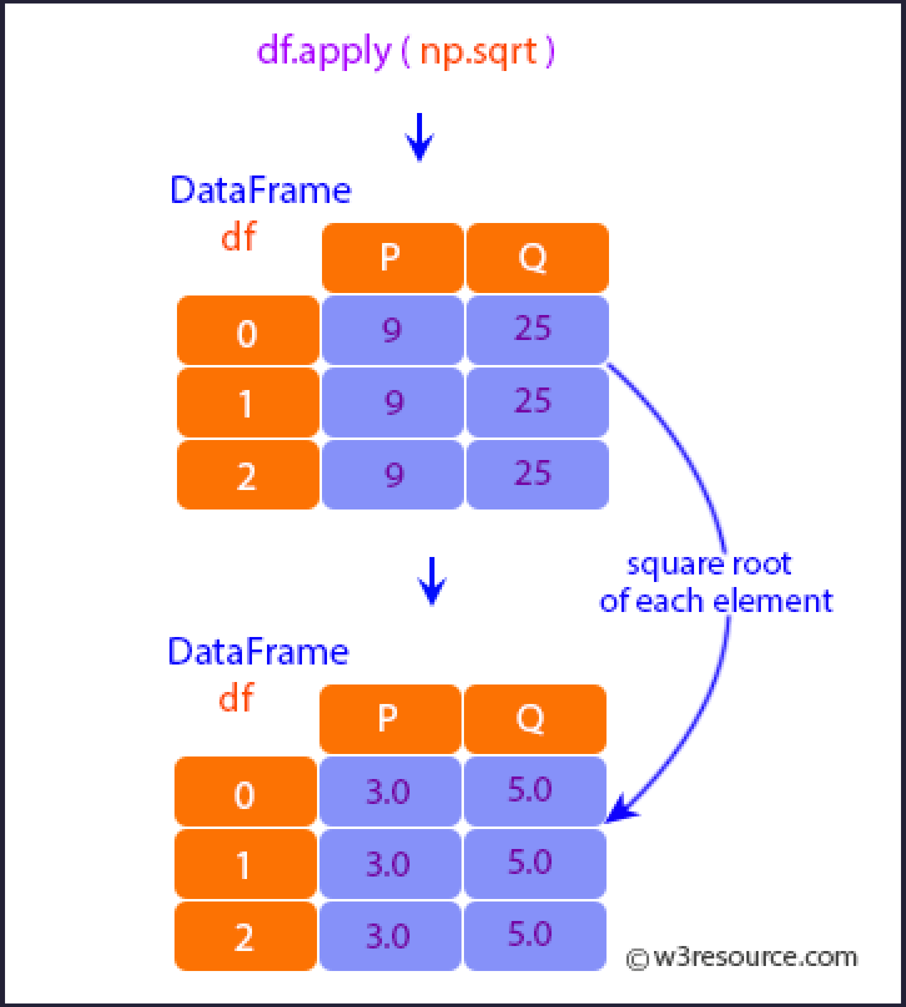
```
response = {
  "resultCount": 1,
  "results": [
    {
      "wrapperType": "track",
      "kind": "feature-movie",
      "collectionId": 1470195095,
      "trackId": 533654020,
      "artistName": "Joss Whedon",
      "collectionName": "Avengers 4-Movie Collection",
      "trackName": "The Avengers",
      ...
    }
  ]
}
```

Navigate through JSON responses

```
response[0]                      # error (not a list)
response.keys()                  # dict_keys(['resultCount', 'results'])
response['results'].keys()       # error (not a dict)
response['results'][0].keys()    # dict_keys(['wrapperType', 'kind', ...])
```

Data wrangling and explorations with Pandas

- Create a DataFrame (`pd.DataFrame`)
- Inspect data (`head`, `info`, `describe`)
- Create new columns (`apply`, `loc`)
- Filter rows (`query`, `loc`)
- Sort rows (`sort_values`)
- Aggregate data (`agg`)
- Group data (`groupby`)
- Join data (`merge`)
- Missing values (`dropna`, `fillna`)



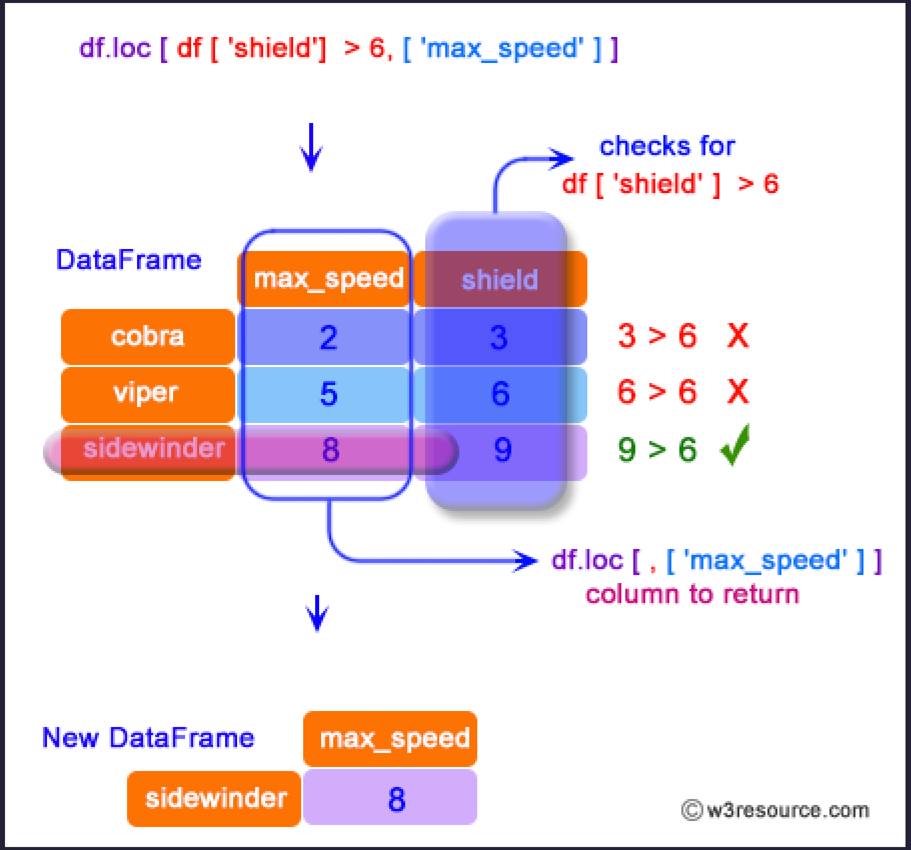
df.apply (np.sqrt)

DataFrame
df

	P	Q
0	9	25
1	9	25
2	9	25

DataFrame
df

	P	Q
0	3.0	5.0
1	3.0	5.0
2	3.0	5.0



df.loc [df ['shield'] > 6, ['max_speed']]

DataFrame

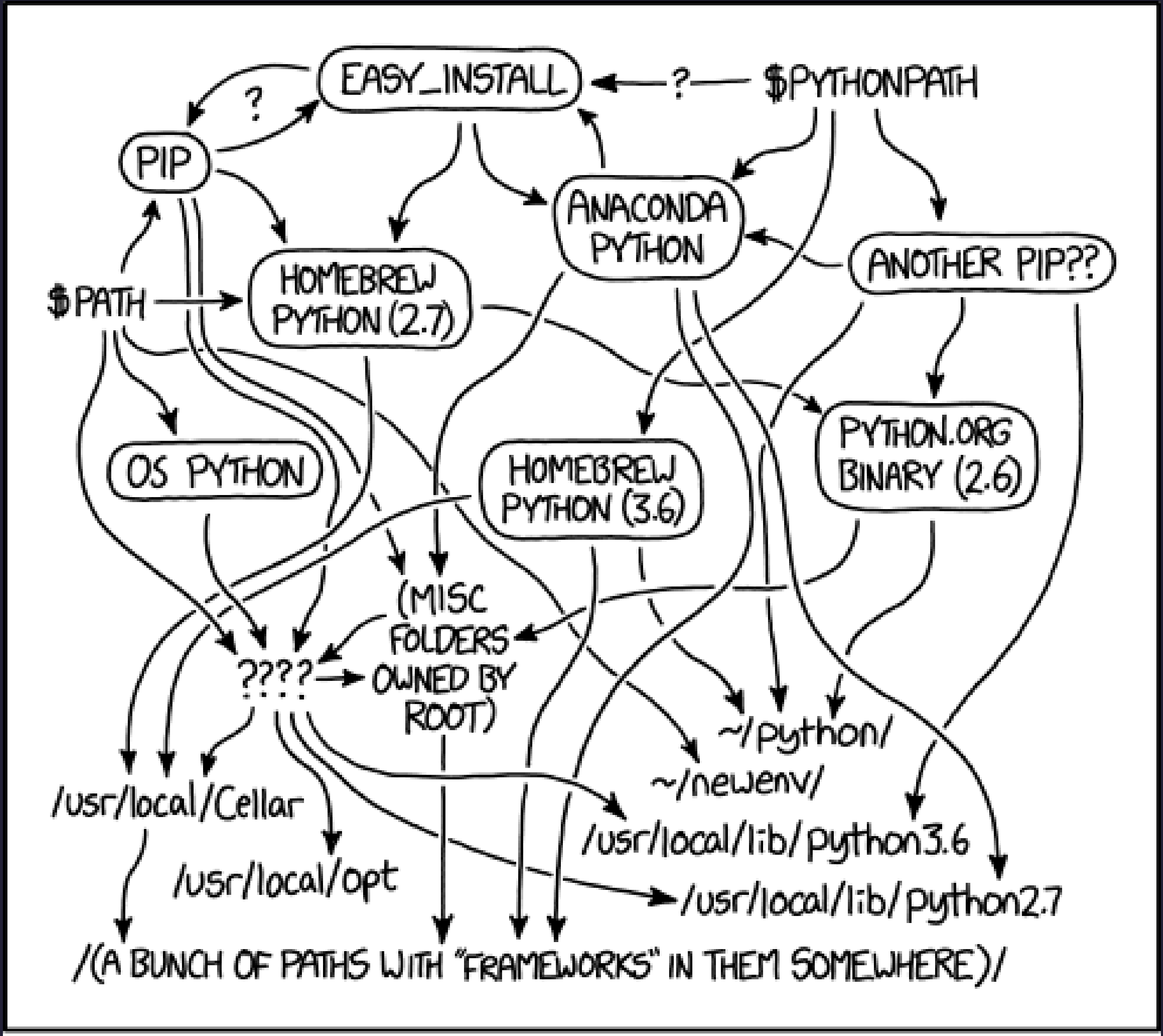
	max_speed	shield	
cobra	2	3	3 > 6 X
viper	5	6	6 > 6 X
sidewinder	8	9	9 > 6 ✓

New DataFrame

	max_speed
sidewinder	8

Final exam logistics

- 2:00 pm on December 10 @ `BRONF 205` & `ARMST 075`
- Exam on EdLesson
- Two parts: 1) multiple choice, 2) coding
- 105 minutes (30+75)
- Closed book, scratch paper allowed
- Cumulative (no visualization)
- Set up your Ed password
- Bring your laptop as a backup
- Make sure `get_my_key()` works with your email address (mock exam)



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Under the hood

- Development environment
- Version control
- Advanced Python topics
- Deployment

Development environment

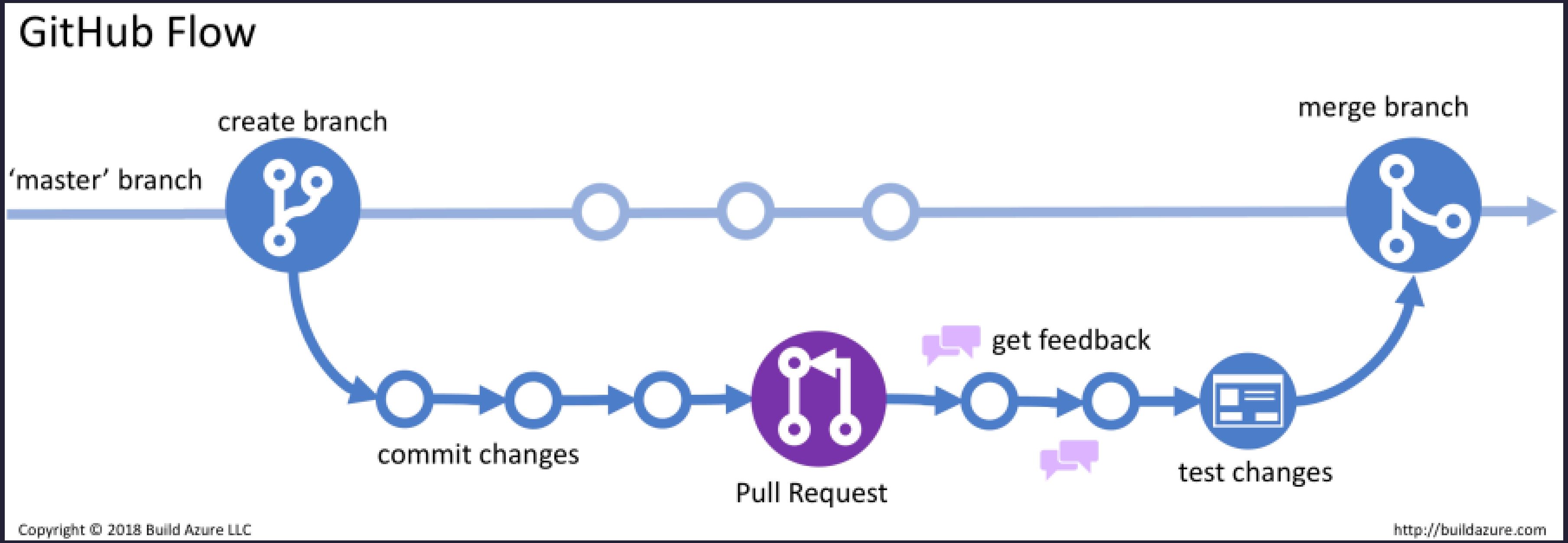
where you write, run, and debug your code

- Cloud: EdLesson, Google Colab (notebook only), GitHub codespaces
- Local
  - Python
  - Code editor (IDE): **VS Code**, PyCharm
  - Command line interface (CLI) and CLI tools
  - Virtual environment (venv, conda, pipenv, poetry)

Version control

Track changes to files

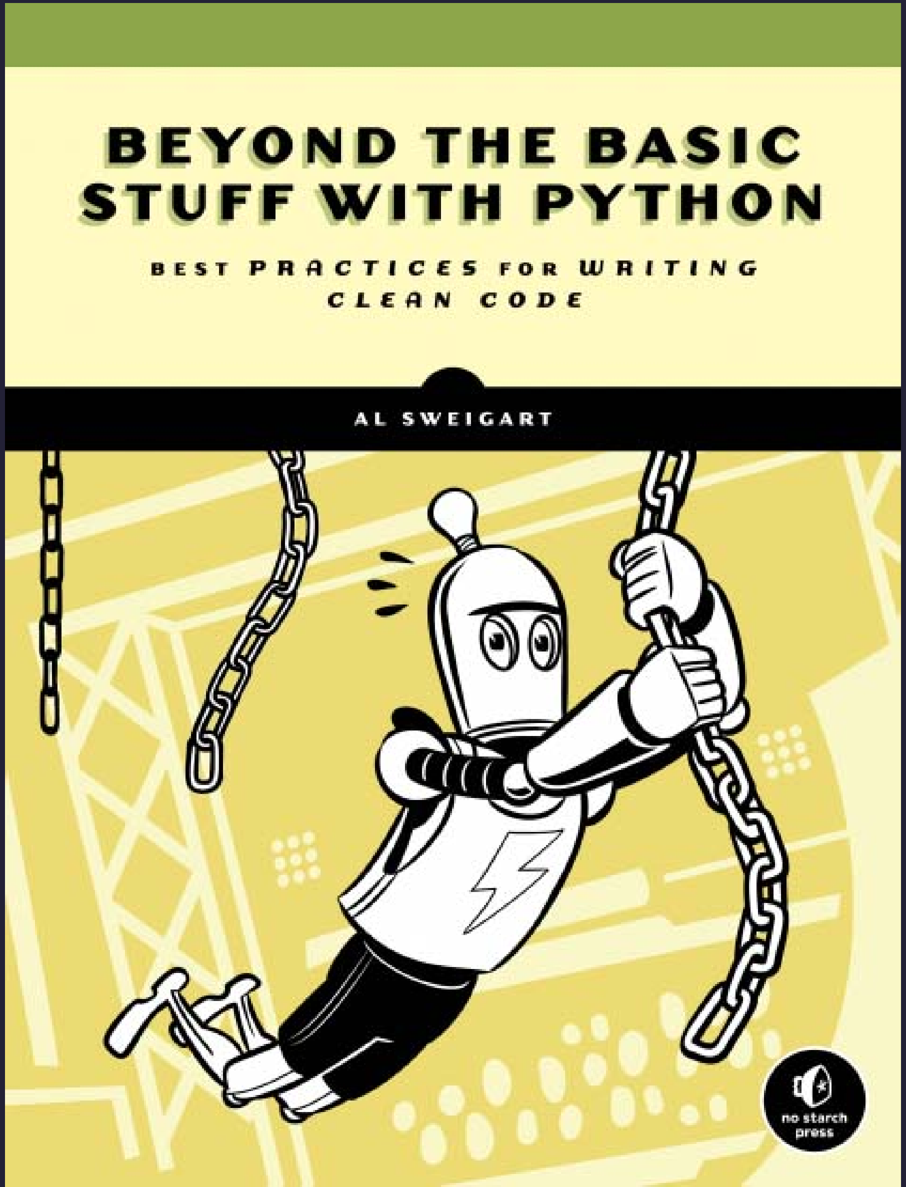
- Git: industry standard protocol for version control
- GitHub: platform for hosting Git repositories



Advanced Python topics

- Object-oriented programming
- Decorators
- Generators
- Testing
- …

https://inventwithpython.com/beyond/


BEYOND THE BASIC STUFF WITH PYTHON
BEST PRACTICES FOR WRITING CLEAN CODE
AL SWEIGART
no starch press

Deployment

- **GitHub repos / GitHub Pages for sharing code**
- **Cloud hosting for sharing apps**
  - General use: Heroku, AWS, GCP, Azure, etc.
  - Data apps: Streamlit, Dash, etc.
  - ML apps: HuggingFace, TensorFlow, PyTorch, etc.
- **Containerization: Docker**
- **DevOps, MLOps, CI/CD**

GitHub Student Developer Pack

- GitHub Codespaces
- **GitHub Copilot**

  …

https://education.github.com/pack

What's next?

- Database: INSY437
- Data mining: INSY446
- Text analytics: INSY448
- Deep learning: INSY463

INSY437: Managing data and databases

- Database design
- Database management systems
- Database administration
- Application development

INSY446: Data mining for business analytics

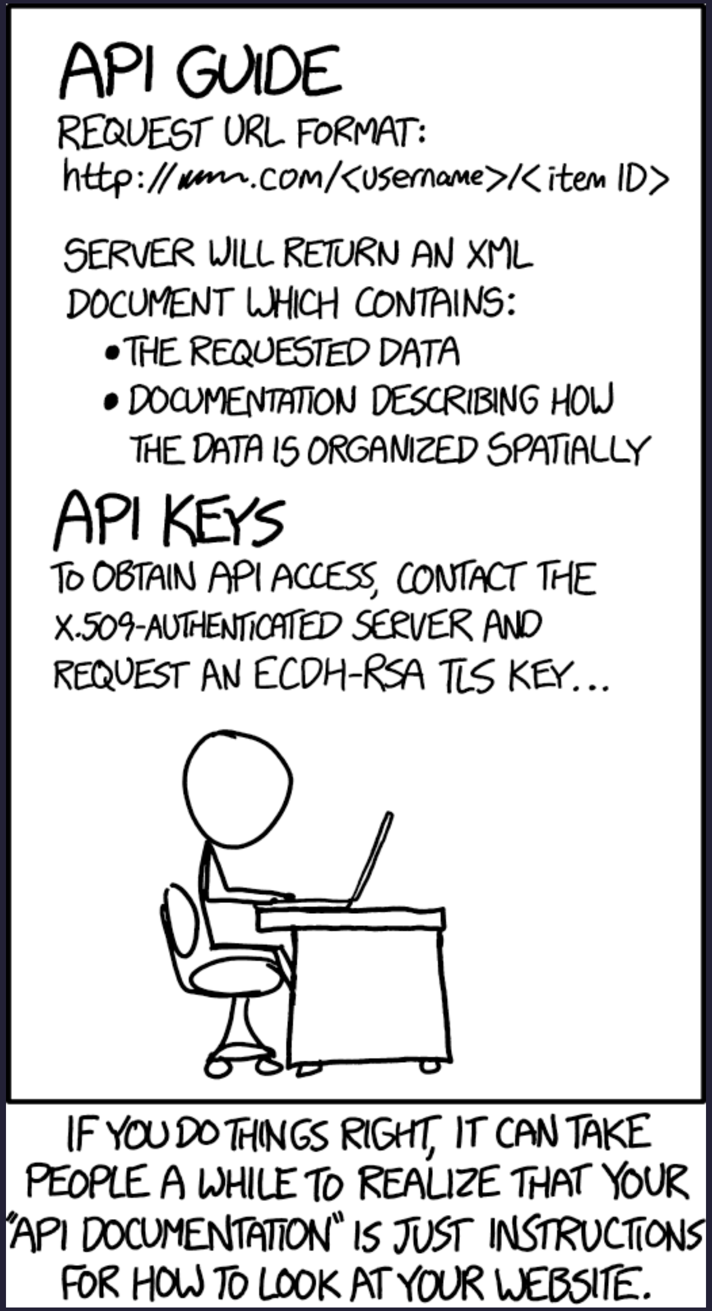
- Advanced data handling (pandas and numpy)
- Supervised learning: Regression and classification
- Neural networks
- Machine learning workflow

INSY448: Text and social media analysis

- Natural Language Processing (NLP) techniques
- Handling text data (tokenization, stemming, lemmatization, etc.)
- Sentiment analysis
- Topic modeling
- Embeddings
- Text summary and classification

INSY463: Deep learning for business analytics

- Advanced neural networks
- Different layer types: CNN, RNN, LSTM, etc.
- Stacking networks
- See https://huggingface.co/tasks for more on what you can do with ML and DL


API GUIDE
REQUEST URL FORMAT:
http://[illegible].com/<username>/<item ID>
SERVER WILL RETURN AN XML DOCUMENT WHICH CONTAINS:
• THE REQUESTED DATA
• DOCUMENTATION DESCRIBING HOW THE DATA IS ORGANIZED SPATIALLY
API KEYS
TO OBTAIN API ACCESS, CONTACT THE X.509-AUTHENTICATED SERVER AND REQUEST AN ECDH-RSA TLS KEY...

IF YOU DO THINGS RIGHT, IT CAN TAKE PEOPLE A WHILE TO REALIZE THAT YOUR "API DOCUMENTATION" IS JUST INSTRUCTIONS FOR HOW TO LOOK AT YOUR WEBSITE.