

# Intro to Python Backends

Day 4: 3DC Introduction to Programming

Raphael Yee

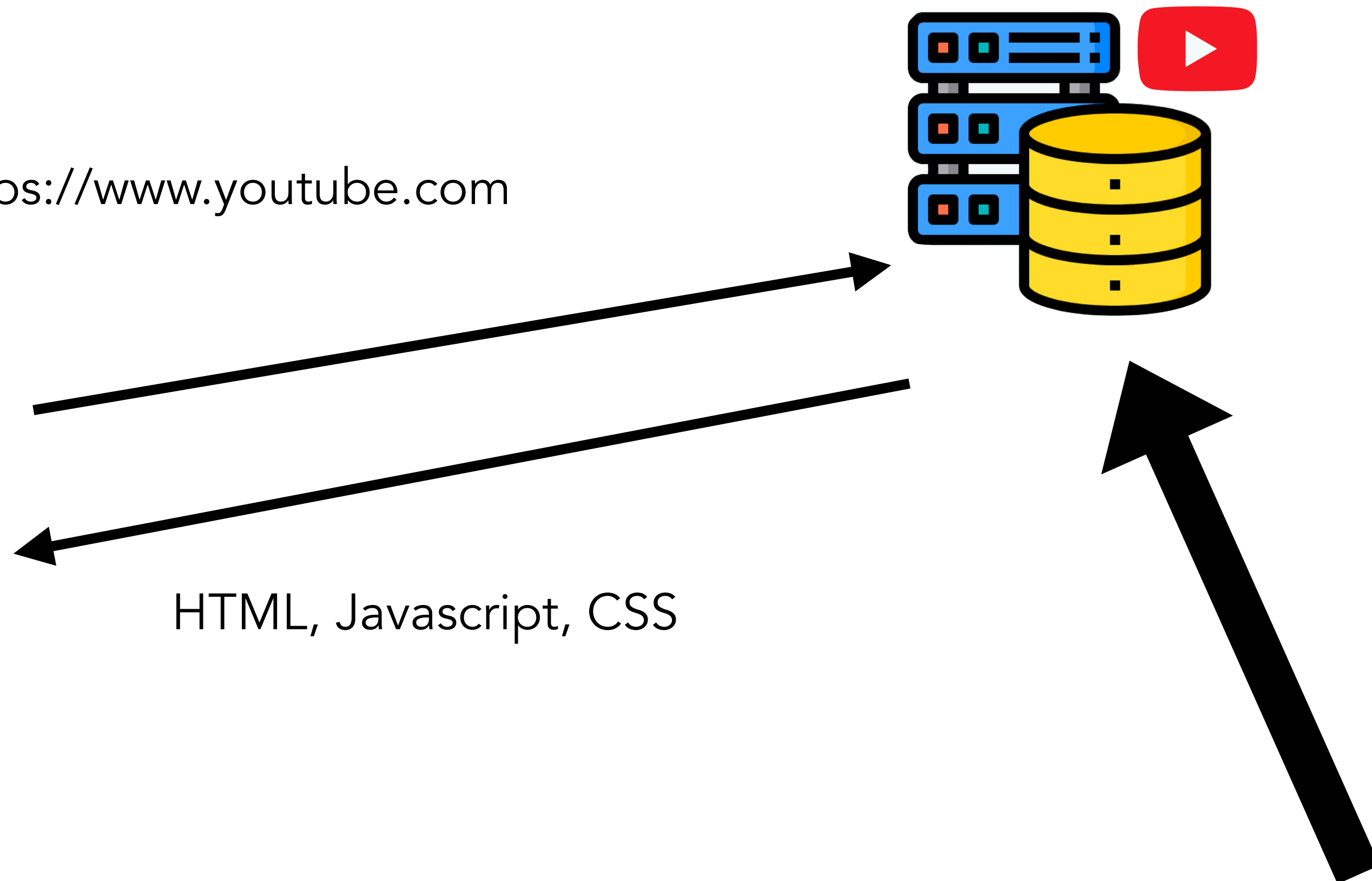
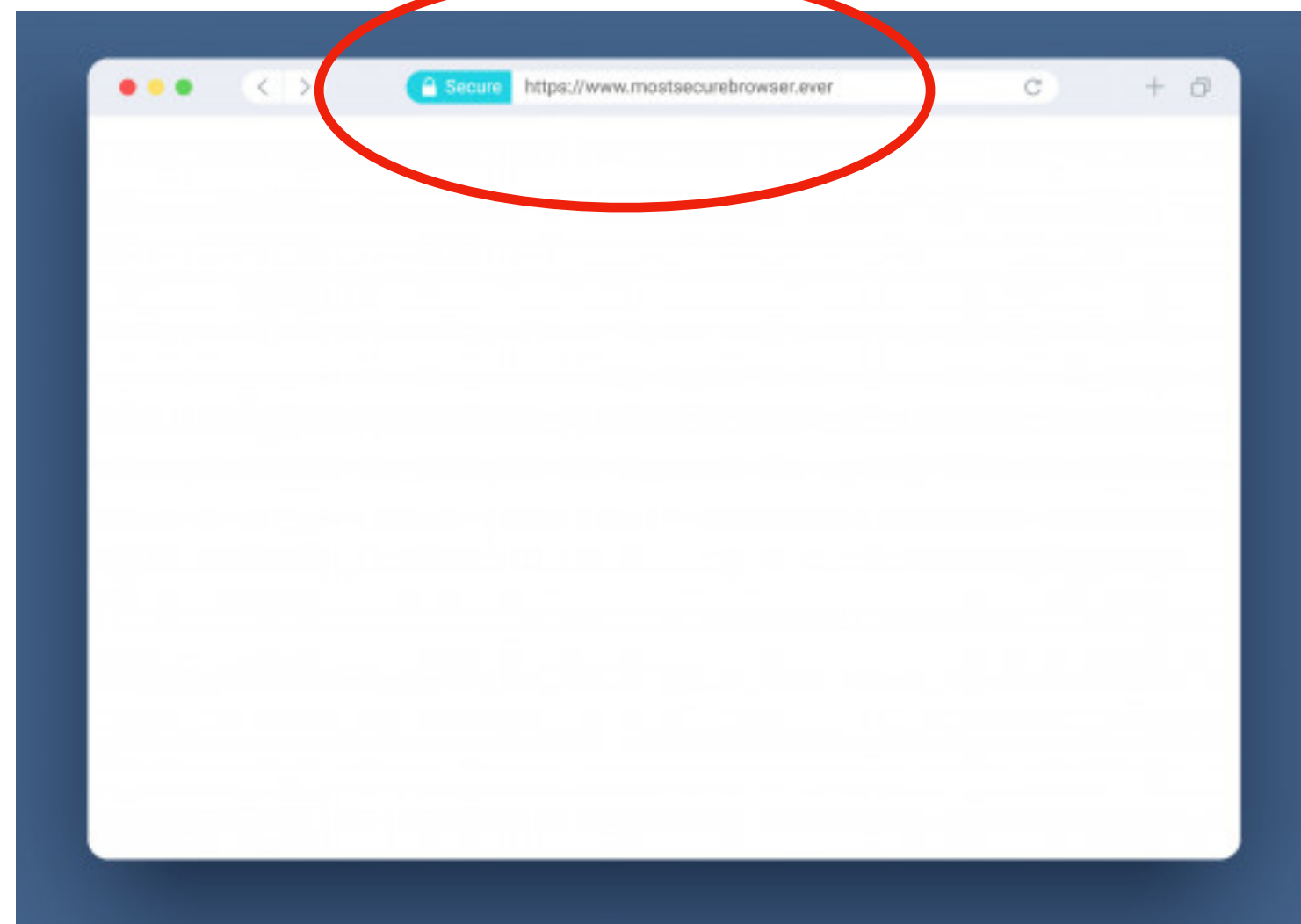
# Recap:

## What have we learnt?

- Information needs to be exchanged for stuff to happen
  - We can do so by making “push” or “pull” requests to/from servers.
  - We can use the Python Requests library to play with data
- ... But what if we want to create the application that responds to these requests?

# Previously...

GET https://www.youtube.com



How does this **process a request** and  
and **return a response**?

# How do application backends work?

How do they parse requests?

Where does information get stored?

How do we retrieve this information?

# How do servers work?

Essentially what's happening is that your app server "listens" on a port for incoming activity.

When it does, it takes the data and does some processing and returns the response.

**Note:** There are different kinds of ways for client and server to "talk"...ultimately, it depends on what your use-case. Again, do you need to push information from both sides or just push from one?

**Think:** What is a **port**? USB-ports, shipping ports...?

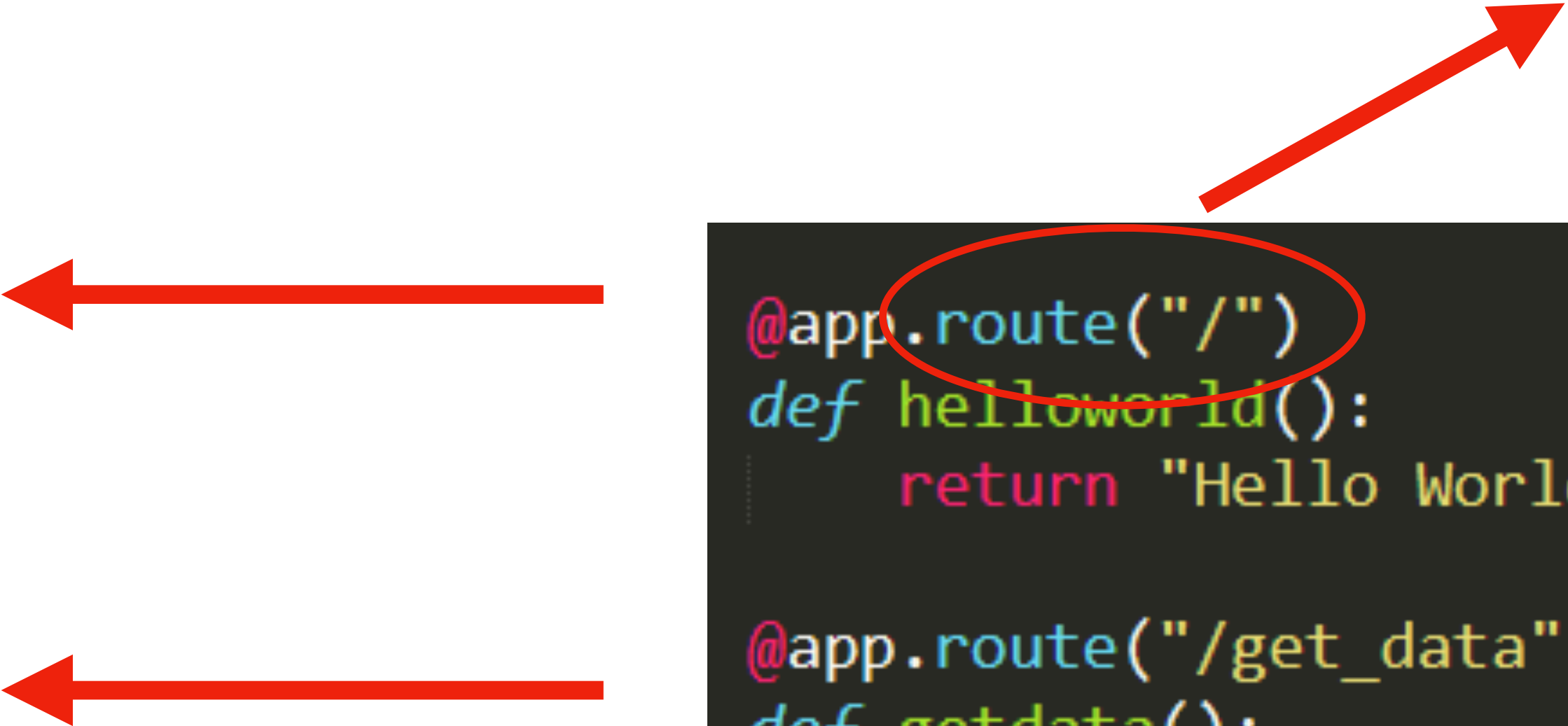


**Read more:** <https://superuser.com/questions/837933/how-do-web-servers-listen-to-ip-addresses-interrupt-or-polling>

## Specifying our API endpoints

GET "/" returns "Hello World!"

GET "/get\_data" returns  
"Your data"

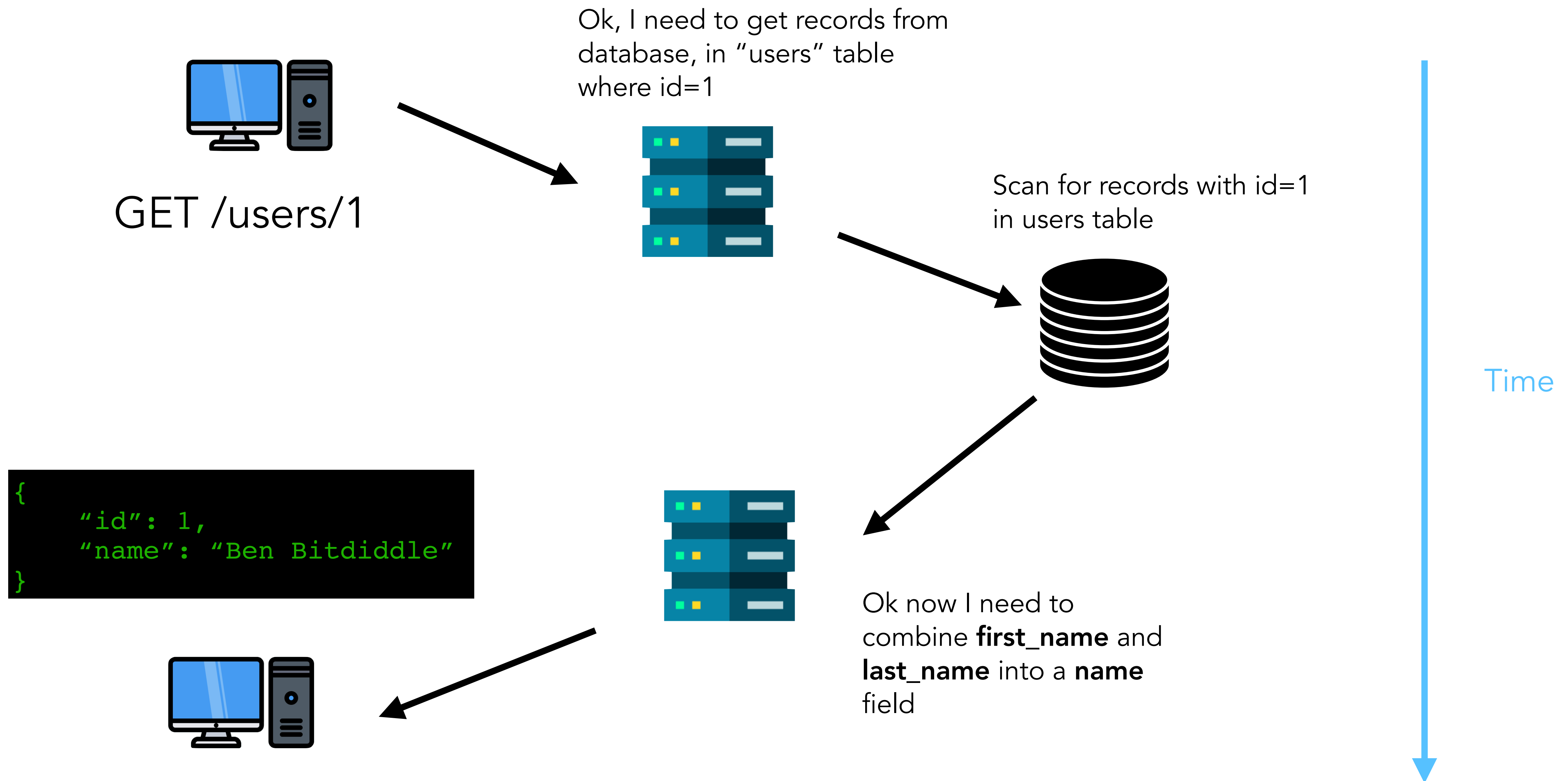


```
@app.route("/")
def helloworld():
    return "Hello World!"

@app.route("/get_data")
def getdata():
    return "Your data"

if __name__ == "__main__":
    app.run()
```

**Think:** In the previous session, we learned that you could return JSON... how will we do that here?



**Think:** What happens when calls are made over the internet? How do they differ from in-process calls?

# What are databases?

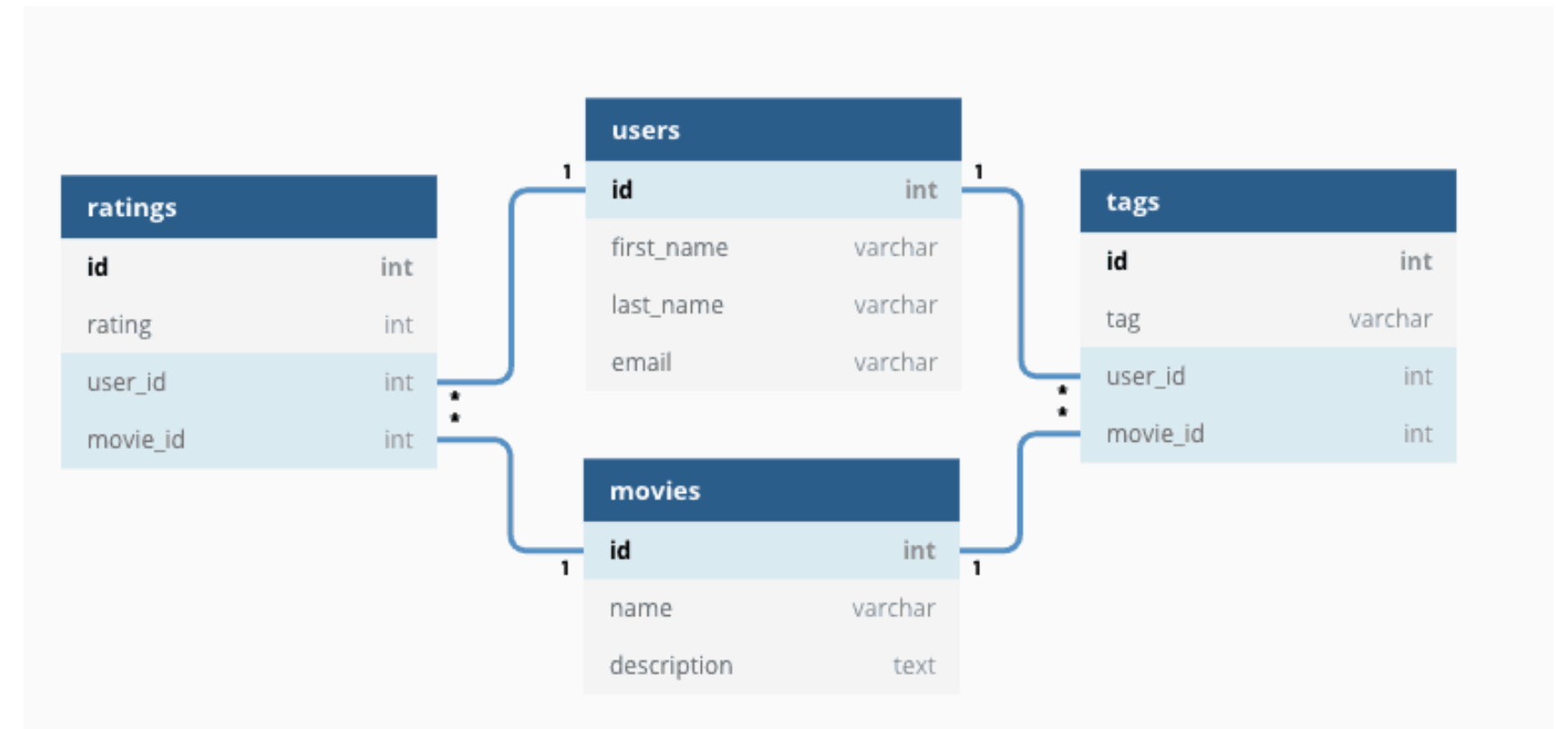
Databases are collections of information that allow you to store, update, delete and manage data.

Basically, if you want to **persist** data after your application ends, you will most likely need a database (or some place to store that data).

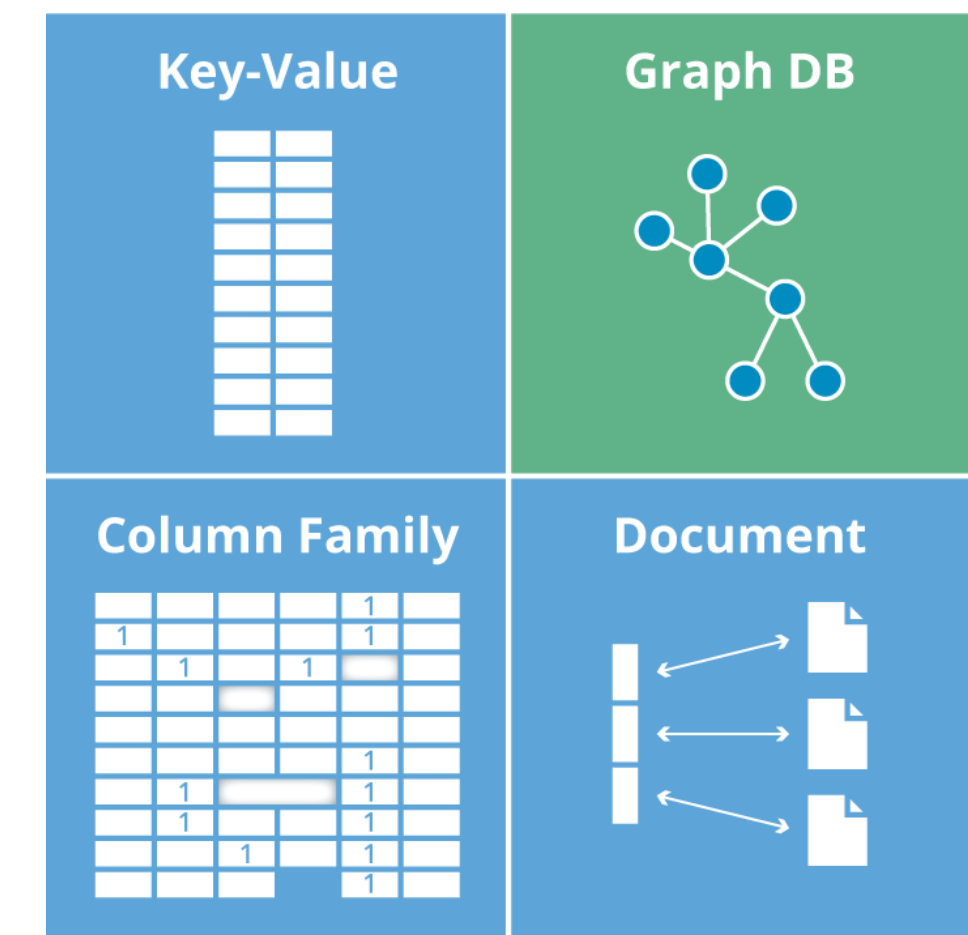
**Note:** There are different kinds of databases for different purposes. You have non-relational and relational databases.

**Think:** What's the difference between a relational and non-relational database?

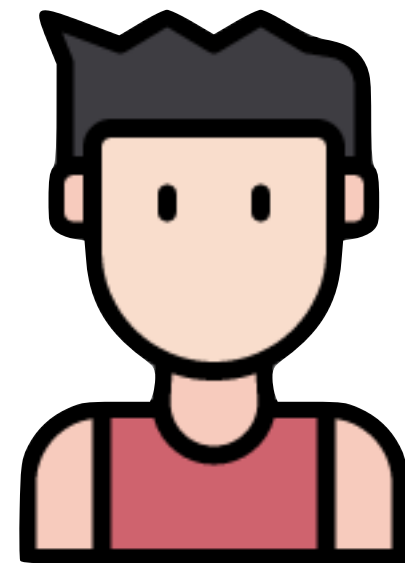
This is a relational database (RDBMS)



These are non-relational types of DBs.



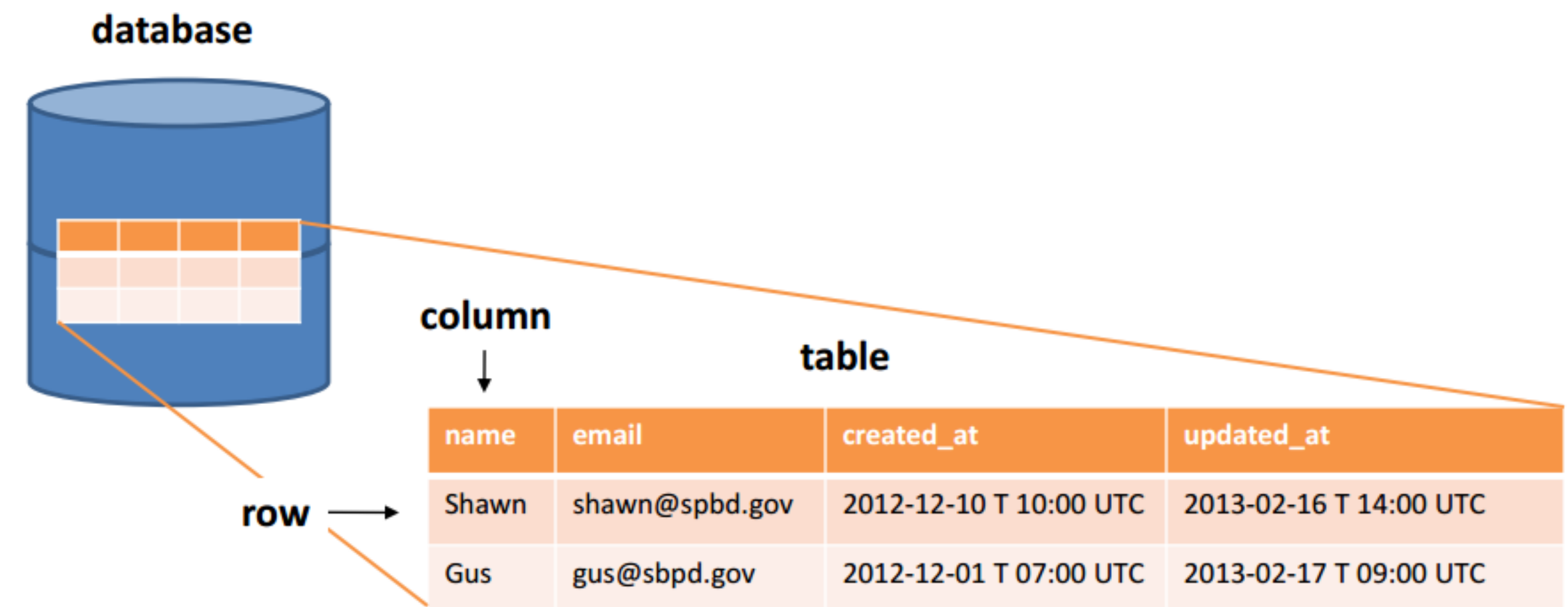




"Hey man, I need the email of every person named Shawn"



```
SELECT
  email
FROM users
WHERE
  name = "Shawn"
```



# Document-based storage

A document database looks very much like JSON. Instead of tables and rows, we have nested “dictionary”-like structures that allow us to query for data using key-values.

**Note:** Notice that the first book has a lot more information than the second book and yet both entries are **valid**. One benefit of document-based storage is that we can define (and similarly leave out) pieces of information we don’t need or don’t have.

In a relational database, your data is very structured and defined by the columns you have (or do not).

**Think:** When do we prefer document-based storage over relational databases?

```
JSON
1  [
2    {
3      "year" : 2013,
4      "title" : "Turn It Down, Or Else!",
5      "info" : {
6        "directors" : [ "Alice Smith", "Bob Jones"],
7        "release_date" : "2013-01-18T00:00:00Z",
8        "rating" : 6.2,
9        "genres" : ["Comedy", "Drama"],
10       "image_url" : "http://ia.media-imdb.com/images/N/09ERWAU7FS797AJ7LU8HN09AMUP908RL1o5JF90EWR7LJKQ7@@._V1_SX400_.jpg",
11       "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
12       "actors" : ["David Matthewman", "Jonathan G. Neff"]
13     }
14   },
15   {
16     "year": 2015,
17     "title": "The Big New Movie",
18     "info": {
19       "plot": "Nothing happens at all.",
20       "rating": 0
21     }
22   }
23 ]
```

# What is JSON?

```
{  
    "id": 1,  
    "name": "Ben Bitdiddle"  
}
```

Javascript Object Notation (JSON) is a text format of data used when sending information from client to server and vice-versa.

You can think of using JSON like Python Dictionaries, both are syntactically similar and use key-value pairs to store/access information.

eg. to print "Ben Bitdiddle": `print(data["name"])`

**Think:** What is the data type of JSON? (strings... integers... etc.)

We'll be creating a quick server using Flask and storing the data on Google Firebase.



Flask



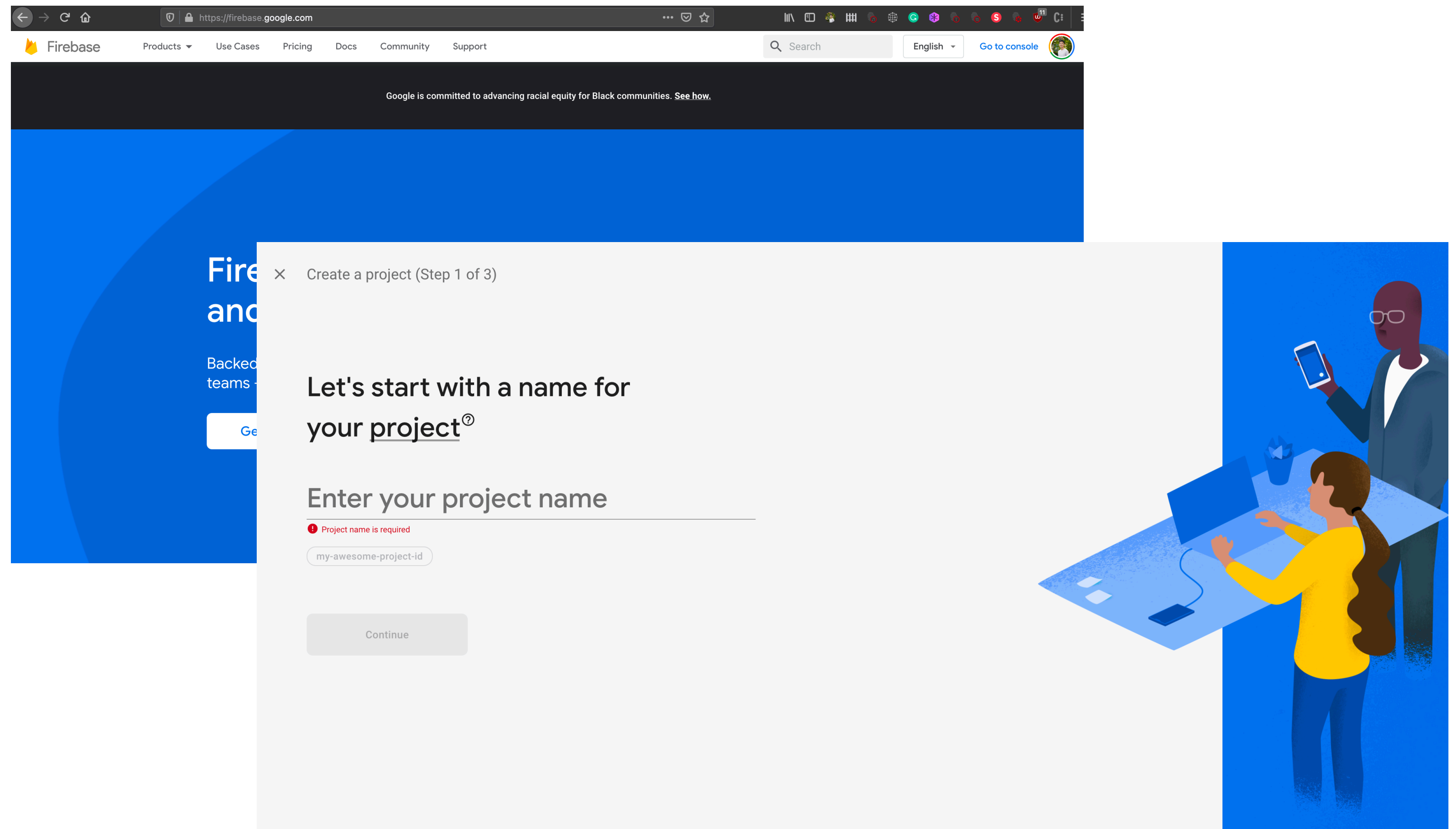
Firebase

# Let's write our first Flask app!

## Activity #1

# Setting up Firebase

Go to [firebase.google.com](https://firebase.google.com)



**Think:** What kind of database is Firebase's Realtime-Database?

# Hello World

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

---

**Think:** Line-by-line, what's happening?

# Let's save your scraped data



# Let's access your scraped data

# Does everything work?

1. Test your functions
2. Does it do what you intended for it to do?
3. Use Postman to test your APIs
4. Any bugs?

# Deploying our app to the web

## Activity #2

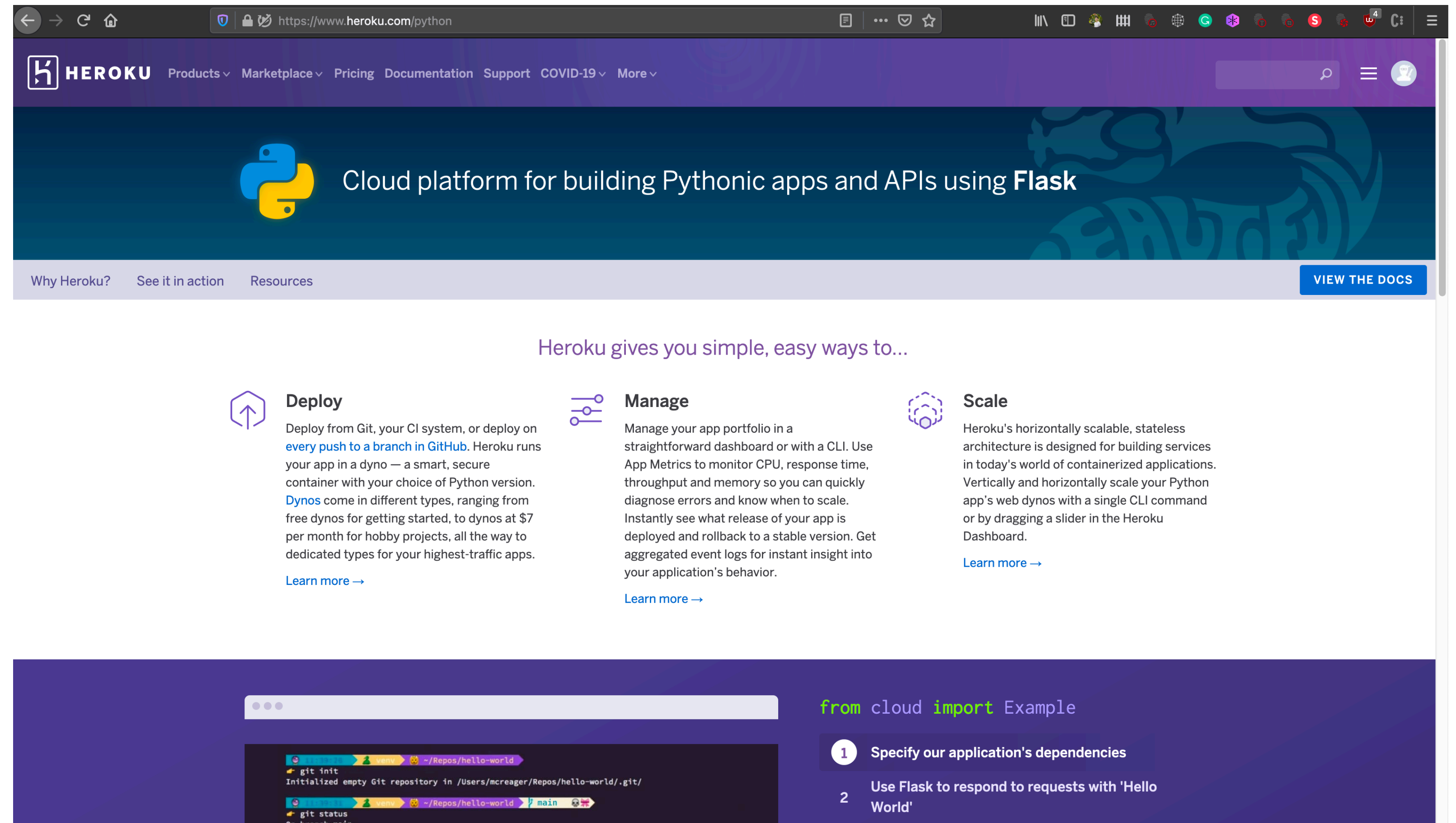
# What is deployment?

Now, your app is available locally on your computer or over the WIFI network. But what if you want it to be available over the internet?

Deployment is the process of putting your app on cloud servers, allowing your app to be accessed over the web.



# Let's go, Heroku!



The screenshot shows the Heroku Python landing page in a web browser. The browser's address bar displays `https://www.heroku.com/python`. The page features a purple header with the Heroku logo and navigation links: Products, Marketplace, Pricing, Documentation, Support, COVID-19, and More. A search bar and a user profile icon are also present. The main content area has a dark blue background with the Python logo and the text "Cloud platform for building Pythonic apps and APIs using **Flask**". Below this, there are links for "Why Heroku?", "See it in action", and "Resources", along with a "VIEW THE DOCS" button. The page is divided into three columns under the heading "Heroku gives you simple, easy ways to...". The first column, "Deploy", includes an icon of an upward arrow and text about deploying from Git or CI systems, mentioning Heroku's dynos and pricing. The second column, "Manage", includes an icon of a gear and text about managing the app portfolio via a dashboard or CLI, mentioning App Metrics and event logs. The third column, "Scale", includes an icon of a cloud and text about Heroku's horizontally scalable, stateless architecture, mentioning vertical and horizontal scaling. At the bottom, there is a code editor snippet showing a terminal window with Git commands and a Python code snippet for a Flask application.

HEROKU Products Marketplace Pricing Documentation Support COVID-19 More

Cloud platform for building Pythonic apps and APIs using **Flask**

Why Heroku? See it in action Resources [VIEW THE DOCS](#)

Heroku gives you simple, easy ways to...

**Deploy**

Deploy from Git, your CI system, or deploy on [every push to a branch in GitHub](#). Heroku runs your app in a dyno — a smart, secure container with your choice of Python version. [Dynos](#) come in different types, ranging from free dynos for getting started, to dynos at \$7 per month for hobby projects, all the way to dedicated types for your highest-traffic apps.

[Learn more →](#)

**Manage**

Manage your app portfolio in a straightforward dashboard or with a CLI. Use App Metrics to monitor CPU, response time, throughput and memory so you can quickly diagnose errors and know when to scale. Instantly see what release of your app is deployed and rollback to a stable version. Get aggregated event logs for instant insight into your application's behavior.

[Learn more →](#)

**Scale**

Heroku's horizontally scalable, stateless architecture is designed for building services in today's world of containerized applications. Vertically and horizontally scale your Python app's web dynos with a single CLI command or by dragging a slider in the Heroku Dashboard.

[Learn more →](#)

```
from cloud import Example
```

```
1 Specify our application's dependencies
2 Use Flask to respond to requests with 'Hello World'
```

Go to [heroku.com](https://heroku.com)

# Recap:

What have we learnt?