Check in,

to claim your free food.

HTML, Flask, SQL

Conquering a Fundamental Web Stack.

First, a word from our sponsor.

Major League Hacking is excited to be a sponsor of HTML, Flask, SQL!

This event is a Pizza Fund 🍕 grant winner.

`hackp.ac/pizzafund-about`

Everything you need this weekend!

hackp.ac/pizzafund-resources

Check your inbox for promo codes 📨

MLH
MAJOR LEAGUE HACKING

.tech
DOMAINS

Free .Tech Domain Name

Register your free domain:

hackp.ac/pizzafund-radix

Event code:

YAYSPRING23

MLH
MAJOR LEAGUE HACKING

GitHub

Join GitHub Global Campus

Gain access to exclusive developer tools like the GitHub Student Developer Pack, which gives you a bunch of **paid resources for FREE.**

hackp.ac/pizzafund-github

MLH
MAJOR LEAGUE HACKING

stackoverflow

Become a Student Ambassador

Build your resume, and get more involved on campus. Earn rewards by planning events like this one!

hackp.ac/pizzafund-stackoverflow

Tweet us @

MLHacks

To get entered to win our monthly Pizza Fund swag raffle.

Tweet must use **#PizzaFund** and **tag MLH.**

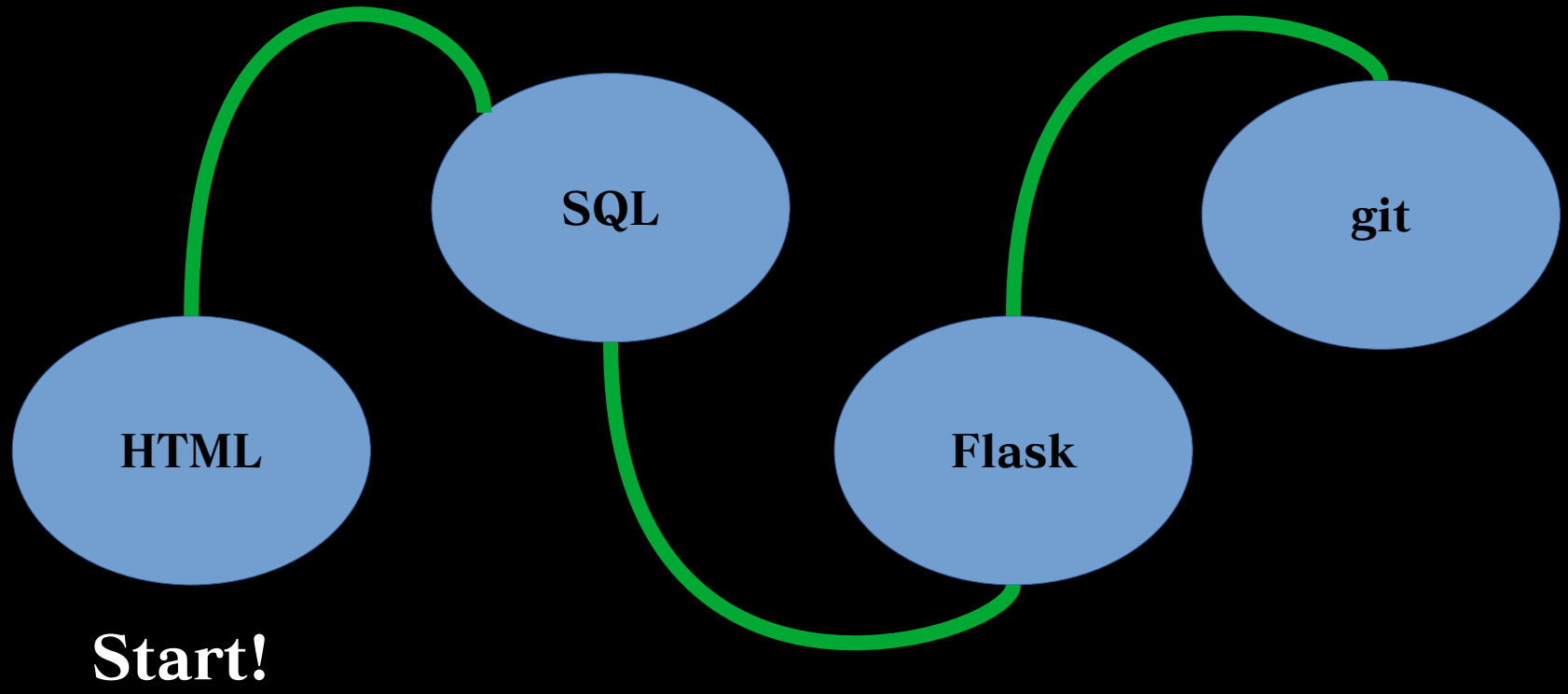
Thanks again to our Pizza Fund Partners!


MLH
MAJOR LEAGUE HACKING

There is a lot to cover, so strap in!

Questions are encouraged!

Roadmap



HTML

SECTION:1

The foundation of every web page.

Hypertext Markup Language

- The language for specifying web pages.
- Tags are the building blocks.
- What you want the page to look like.

```
<tag>content</tag>
```

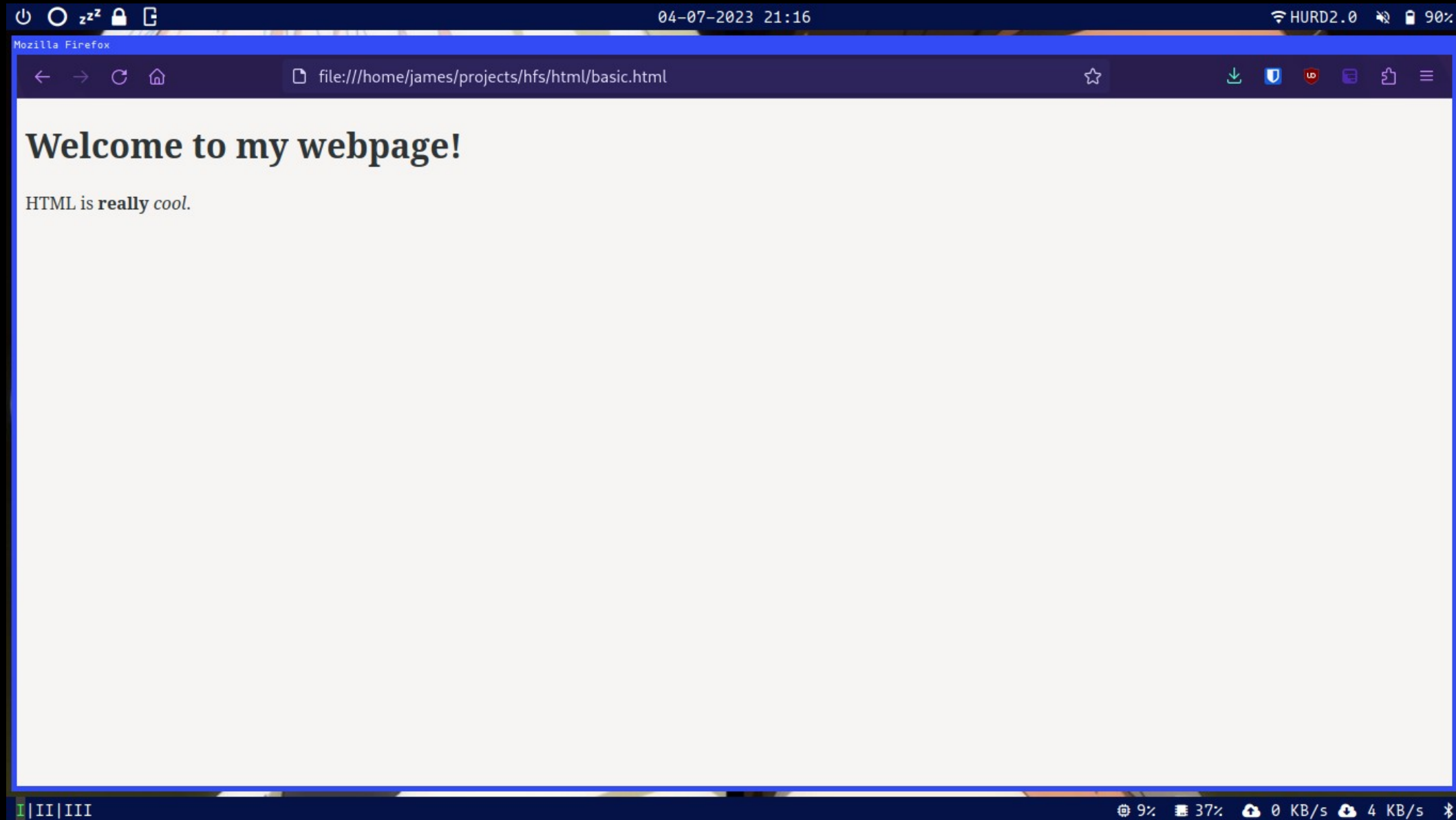
```
<!DOCTYPE html>
<html>
  everything lives here
</html>
```

```
<!DOCTYPE html>
<html>
  <body>
     main content of web page
  </body>
</html>
```

Basic Tags

- `<p>` -- **Paragraph.**
- `<h1>` -- **Header.**
- `<h2>` -- **Subheader.**
- `<h3>` -- **Subsubheader.**
- `<h4>` -- **You get the idea... (goes down to 6).**
- `<strong>` -- **Bold text.**
- `<em>` -- ***Italic text.***
- `<br>` -- **Line break (no content => does not need end tag).**

```
<!DOCTYPE html>
<!-- This is a comment. -->
<html>
 <body>
    <h1>Welcome to my webpage!</h1>
    <p>
     HTML is <strong>really</strong> <em>cool</em>.
    </p>
 </body>
</html>
```


04-07-2023 21:16
HURD2.0
90%
Mozilla Firefox
file:///home/james/projects/hfs/html/basic.html
Welcome to my webpage!
HTML is really cool.
I|II|III
9%
37%
0 KB/s
4 KB/s

HTML Lists

- Can be ordered or unordered.
- `<ul>` -- Unordered list (e.g. bullets).
- `<ol>` -- Unordered list (e.g. 1, 2, 3...).
- `<li>` -- Item in both lists.

```
<!DOCTYPE html>
<html>
 <body>
  <h1>Unordered List.</h1>
  <ul>
   <li>item one</li>
   <li>item two</li>
  </ul>
 </body>
</html>
```

```
<!DOCTYPE html>
<html>
 <body>
 <h1>Ordered List.</h1>
  <ol>
   <li>item one</li>
   <li>item two</li>
  </ol>
 </body>
</html>
```

Unordered List.

- item one
- item two

Ordered List.

1. item one
2. item two

HTML Tables

- `<table border="border_size">` -- Table container.
  - Border size can be many things:
    - e.g. px, em, pt
- `<tr>` -- Table row container.
- `<th>` -- Table heading.
- `<td>` -- Table cell contents.

```
<!-- Outer tags omitted for readability. -->
    <table border="5pt">
      <tr>
        <th>Name</th>
        <th>Flavor</th>
      </tr>
      <tr>
        <td>Ethan</td>
        <td>Cookies and Cream</td>
      </tr>
      <tr>
        <td>Sandy</td>
        <td>N/A</td>
      </tr>
</table>
```

My Friends' Favorite Ice Cream Flavors

Ranking	Flavor Name
Ethan	Cookies and Cream
Sandy	N/A

Ranking	Flavor Name
Ethan	Cookies and Cream
Sandy	N/A

<table>

<td>

<th>

<tr>

HTML Links

- `<a href="link destination">` -- **Hyperlink.**
- **Link destination could be many things:**
  - **Local HTML page.**
    - **e.g.** `"path/to/page.html"`
  - **Other things on the web.**
    - **e.g.** `"https://youtu.be/dQw4w9WgXcQ"`

```
<!-- Outer tags omitted for readability. -->

<h1>Links!</h1>

<ul>
    <!-- basic.html is another page in the current directory. -->
    <li><a href="basic.html">Check out my awesome page!</li>
    <li><a href="https://youtu.be/dQw4w9WgXcQ">Look at this cool video.</li>
</ul>
```

Links!

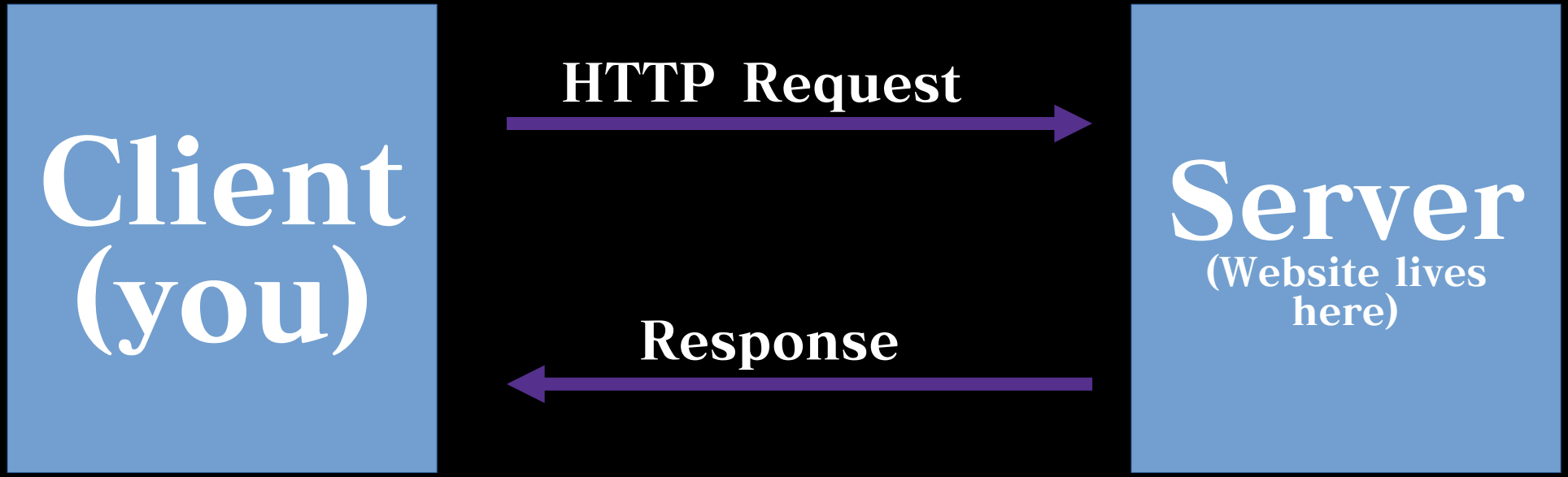
- Check out my awesome page!
- Look at this cool video.

Welcome to my webpage!

HTML is **really** *cool*.

Hypertext Transfer Protocol

- How you actually talk with the web.
- Many flavors of requests.



GET Requests

- "Get" the state of the server.
- Retrieve data, no changes.

POST Requests

- Hand new data to the server.
- Useful in form responses, file uploads.

HTML Forms – Container and Actions

- `<form method="http method" action="where to send data">`
  - `method` -- **HTTP method to send data with.**
    - **GET is default, appends data to new URL and goes there.**
      - **e.g.** `myform.com` → `myform.com?response=form_response`.
    - **We will use POST, which sends data in POST request.**
  - `action` -- **Where form sends data, default is to current page.**
    - **We will use the default, but could use PHP, Web API, etc.**

HTML Forms – Input and Labels

- The `<input>` tag can take many forms, a couple are:
  - `<input name="name">` -- Text input identified by `name`.
    - Add `required` to make it a required input.
      - e.g. `<input name="input_name" required>`
  - `<input name="submit" value="text">` -- From submit button.
    - `text` will appear on the button.
  - `<input>` tags do not require an end tag.
- `<label for="name">` -- Label for input identified by `name`.

```
<!-- Outer tags omitted for readability. -->
<h1>Innocent Form</h1>
    <form method="post">
      <label for="ssn">Social Security #</label>
      <input name="ssn" required>
      <input type="submit" value="Press here for free stuff!">
    </form>
```

POST Request

ssn=785-330-9797

What do we do with this request? Just wait…

Still with me?

Questions? Speak now or forever hold you peace!

SQL

SECTION:2

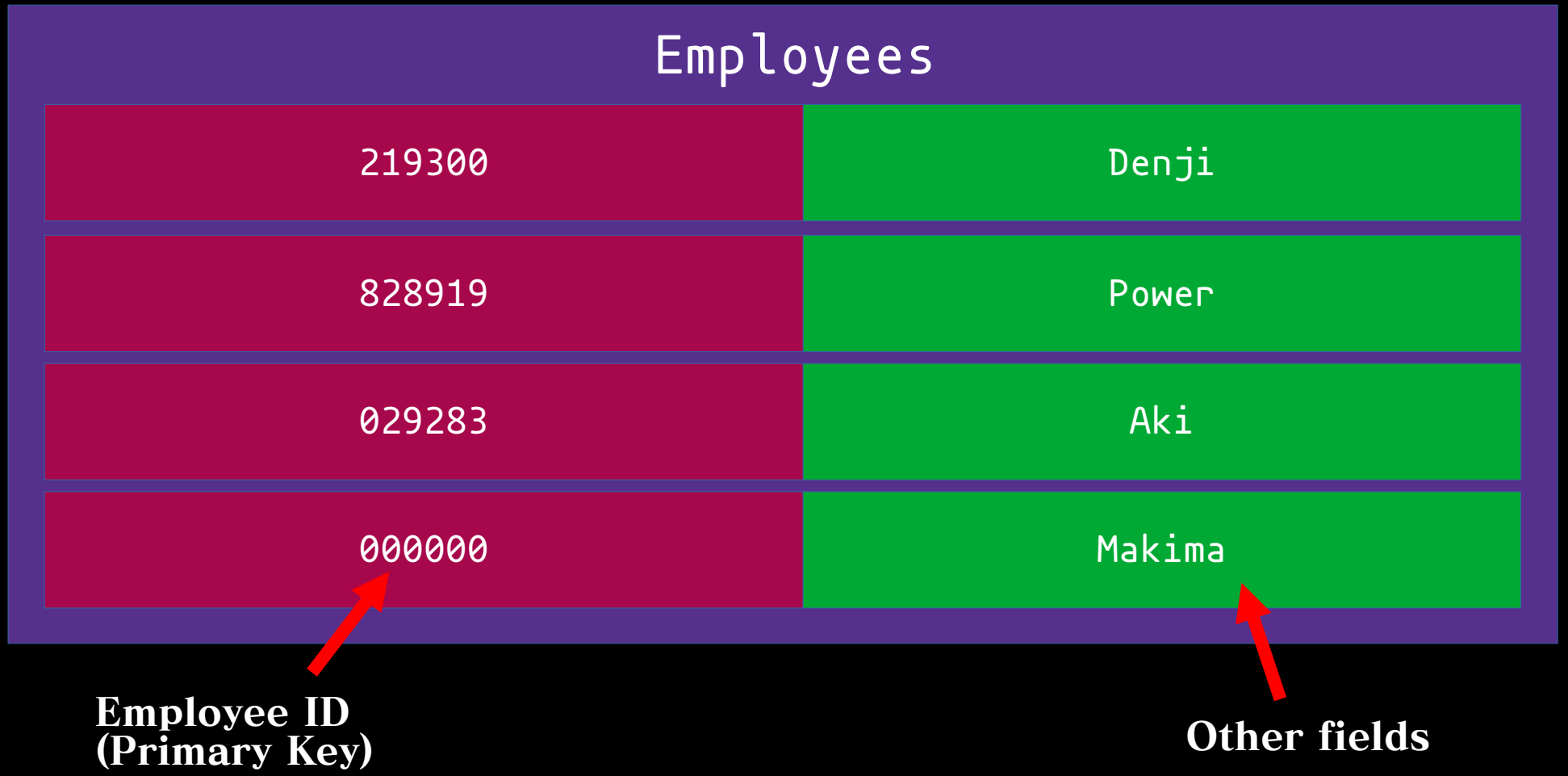
Persistently storing data.

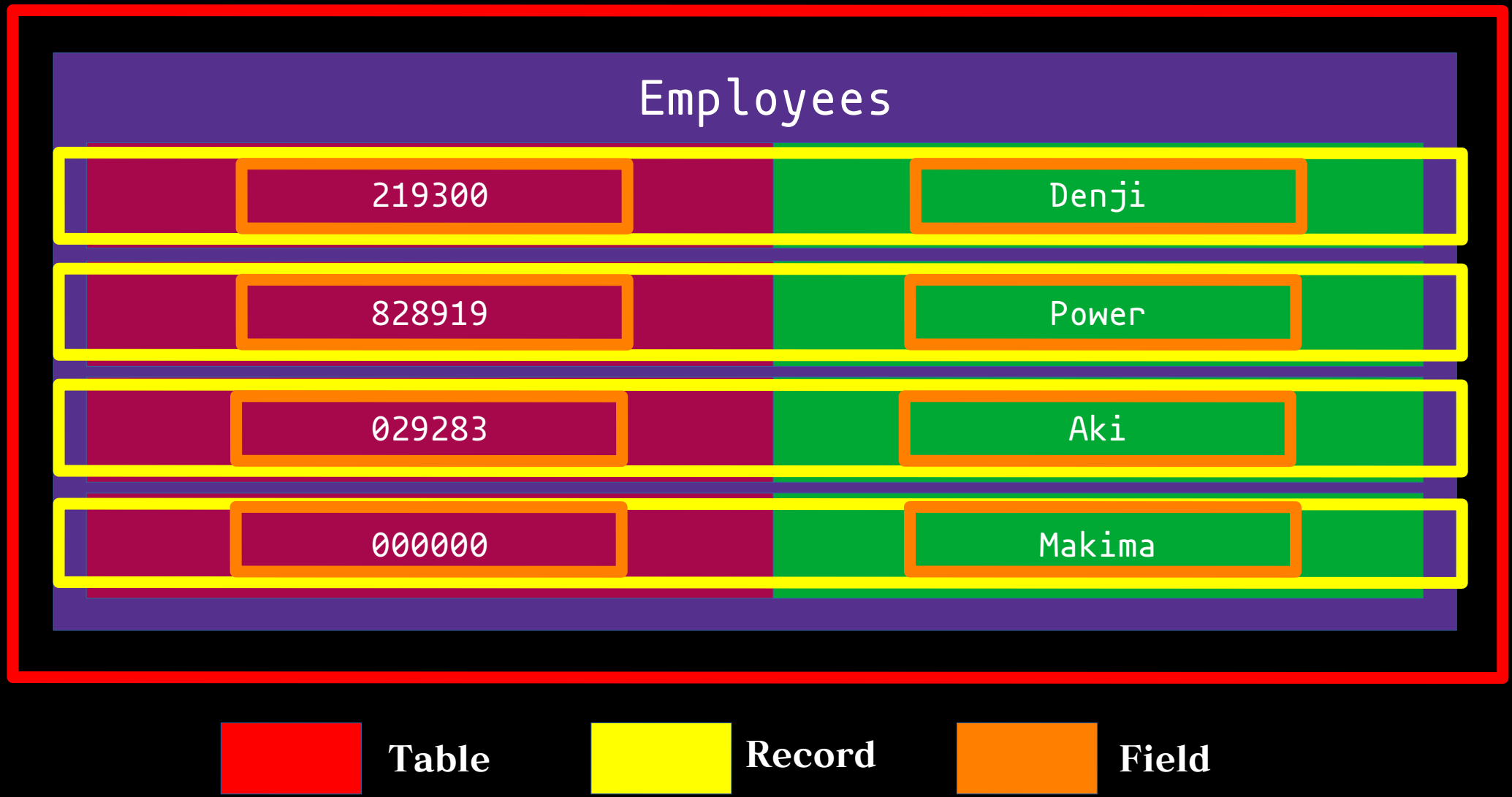
What is a Database?

- A persistent way to store and access data.
  - Could be as simple as a `txt` file.
- Two main flavors:
  - Relational (e.g. SQLite)
    - We will focus on these.
  - Non-relational (e.g. MongoDB)

What is a Relational Database?

- Data is organized into tables.
- Each table is rows and columns
- A unique key identifies each row.
  - Called the “primary key”.
- Sort-of like a spreadsheet.


Employees
219300
Denji
828919
Power
029283
Aki
000000
Makima
Employee ID
(Primary Key)
Other fields


Employees
219300
Denji
828919
Power
029283
Aki
000000
Makima
Table
Record
Field

Database Management System (DBMS)

- How you interact with the database.
- Allows you to CRUD, or
  - Create
  - Read
  - Update
  - Delete
- We will use SQLite, but there are many options:
  - e.g. MySQL, MariaDB.
- Specifically, we will use an RDBMS

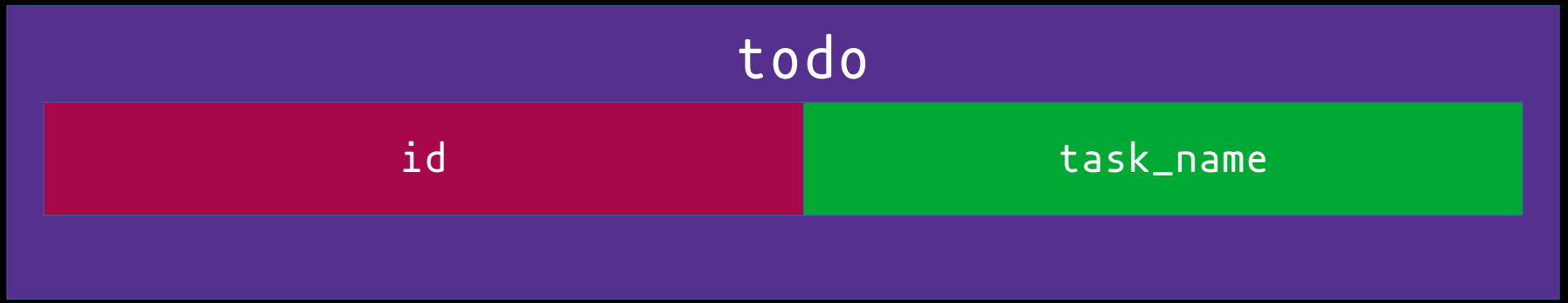
Structured Query Language

- Standard language for interacting with an RDBMS.
- Nonstandard across different RDBMS.
  - Basics are more or less the same everywhere.
- Use “statements” to do things to database.
  - End with a semicolon.
  - Not case-sensitive.
- Input statements:
  - Directly at RDBMS prompt.
  - In a `.sql` file.

Creating/Deleting a Table

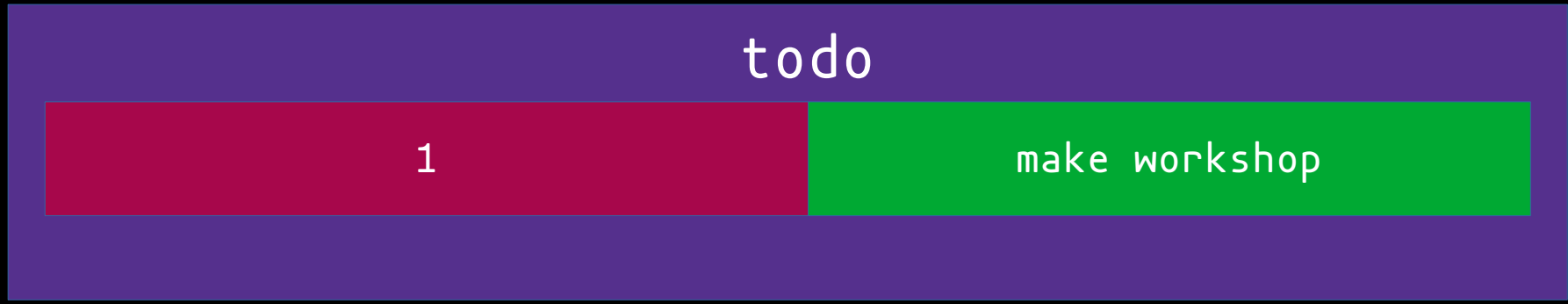
- **We create a table by specifying a *Schema*.**
  - **Schema => What a typical record looks like.**
- `CREATE TABLE name(f1 datatype, f2 datatype, …);`
  - `CREATE TABLE IF NOT EXISTS` **is another way to do this.**
- **Fields can be many types.**
  - **e.g. integer, text, float**
- **Each field can also have special options after it.**
  - `AUTOINCREMENT` **is useful for ints.**
    - **Automatically generates new int on insert.**
- `DROP TABLE name;` **-- Deletes table "name".**

```
CREATE TABLE todo(
  id INTEGER AUTOINCREMENT,
  task_name TEXT
);
```

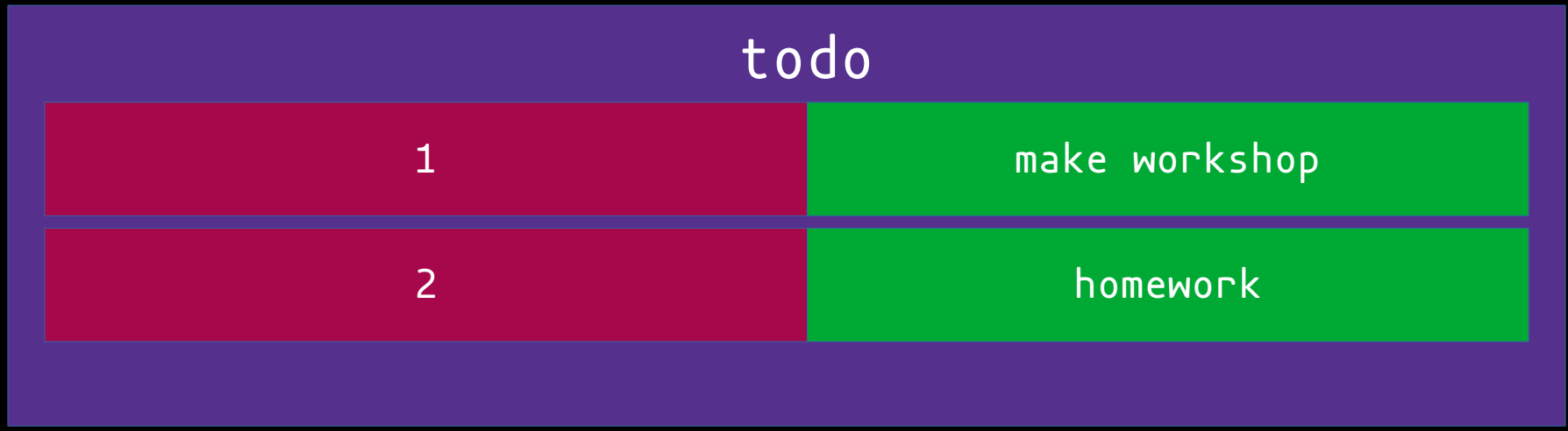


Messing With Records

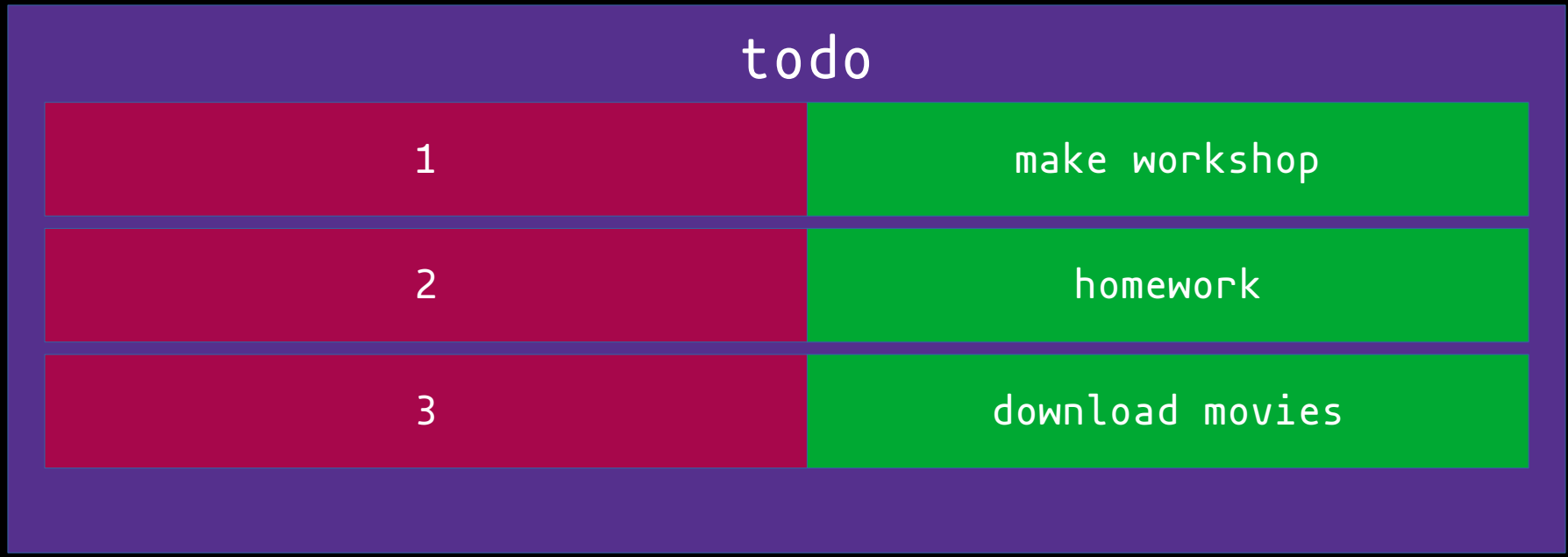
- `INSERT INTO name VALUES (f1_value, f2_value, ...);`
  - **Creates a new record with given field values in table** `name`**.**
  - **Fields with** `AUTOINCREMENT` **will be automatically created.**
- `DELETE FROM name WHERE condition;`
  - **Delete all records where** `condition` **is true.**
  - **No** `WHERE` `=>` **all records deleted!!!!!**
- `UPDATE name SET f1=newval1, f2=newval2, ... WHERE condition;`
  - **Changes** `f1` **to** `newval1`, `f2` **to** `newval2`, **etc. where** `condition` **is true.**
- `Condition` **can be any expression that evaluates to a bool.**
  - **e.g.** `field = value`, `field < value`**.**



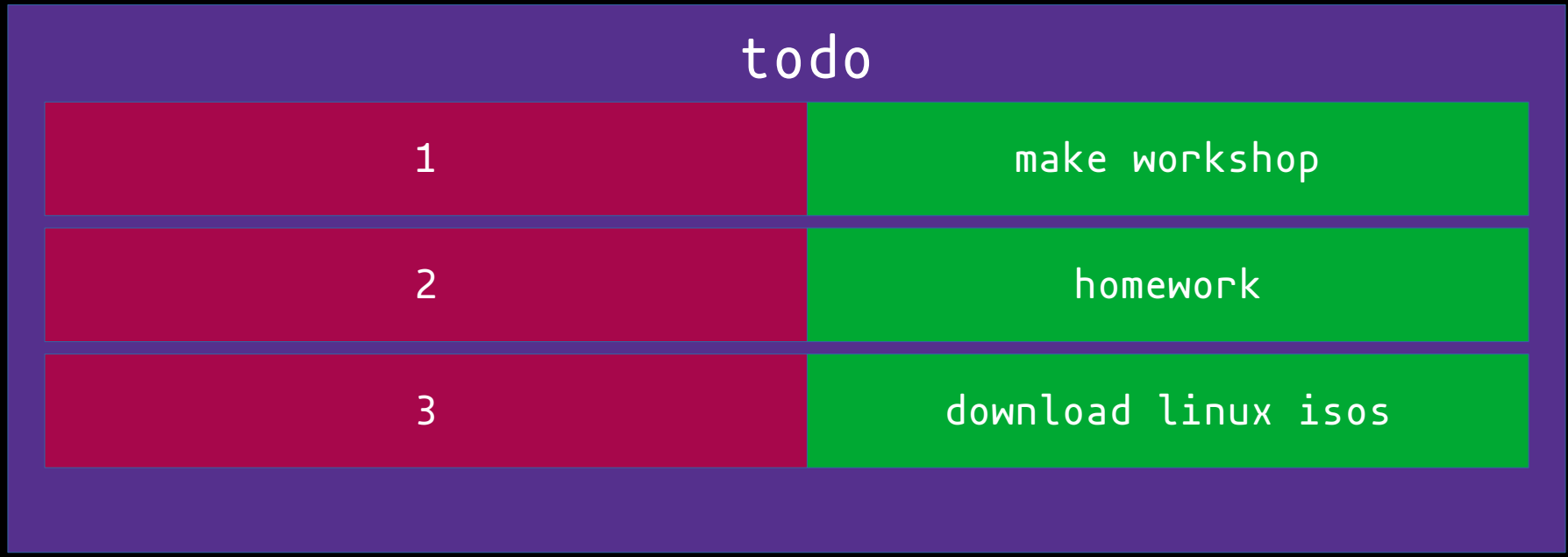

```
INSERT into todo VALUES (make workshop);
```



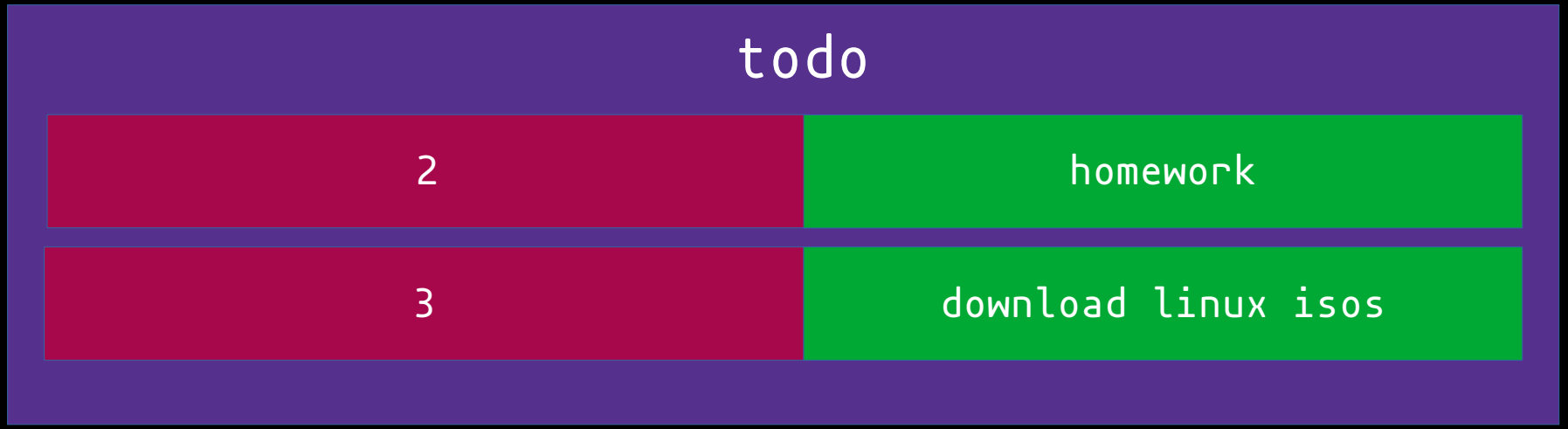

```
INSERT into todo VALUES (homework);
```




```
INSERT into todo VALUES (download movies);
```



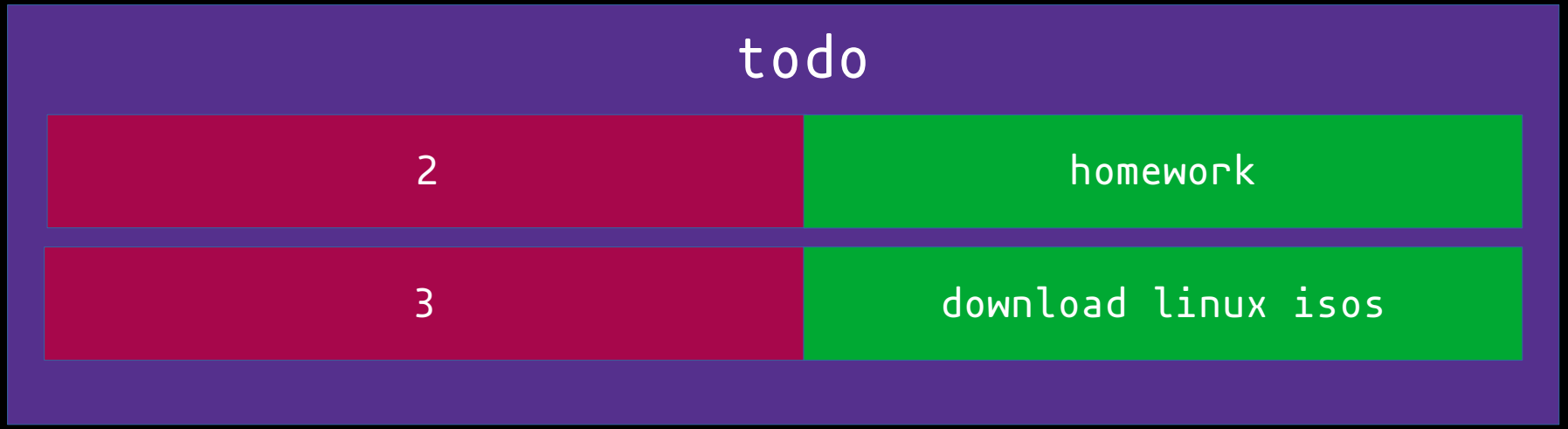

```
UPDATE todo SET task_name='download linux isos' WHERE id=3;
```




```
DELETE from todo WHERE id=1;
```

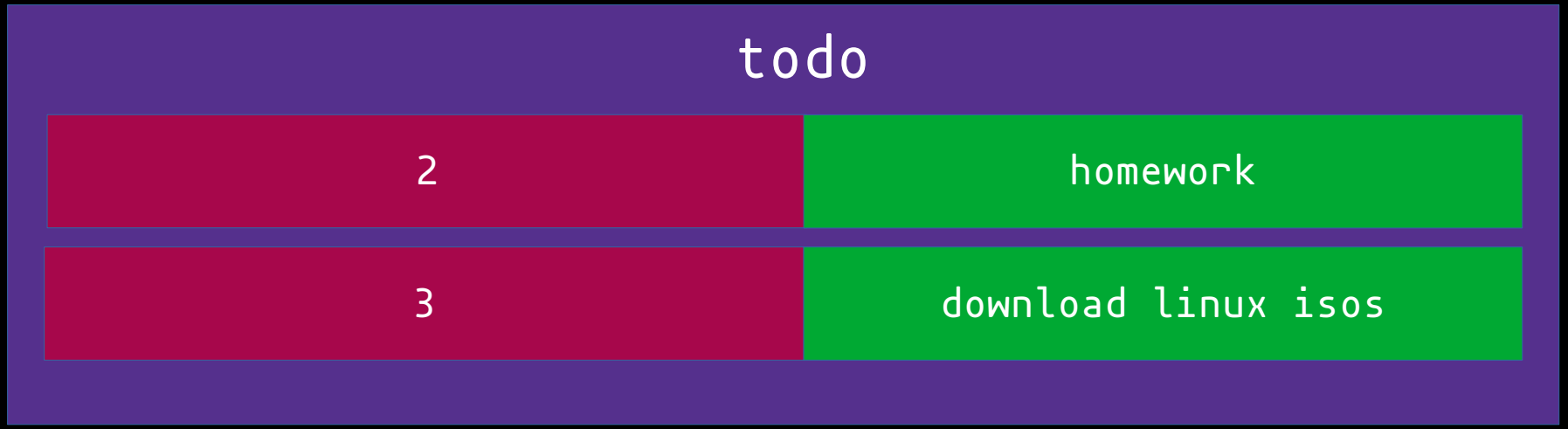
Getting Records

- **Querying == getting records form a database.**
- `SELECT * from name WHERE condition;`
  - **Returns all records from `name` where `condition` is true.**
- `SELECT f1, f2, ... from name WHERE condition;`
  - **Returns fields `f1, f2, ...` from `name` where `condition` is true.**
- `condition` **is the same as before.**




```
SELECT * from todo WHERE id=2;
```

```
(2, homework)
```




```
SELECT task_name from todo WHERE id=2;
```

```
(homework)
```

Everyone Still Onboard?

I know this is a lot of information. Please ask questions!

Okay, this is all cool but how do we actually use these things?

Flask

SECTION:3

Putting it all together.

What is a Web Framework?

- A collection of tools to build and deploy web applications.
- Manages:
  - Web server connection.
  - Database connection.
  - Templating.
  - Much more!

A Brief Note on Decorators

- What they actually are isn't important.
- Adds extra functionality to our functions.
- We will only import these from Flask.
- For nerds: Decorators == higher order functions.

@

A Brief Note on Decorators

```
#!/usr/bin/python

@do_twice
def hello_world():
    print("hello, world!")
hello_world()
```

```
hello, world!
hello, world!
```

What is Flask?

- A web framework written in Python.
- Get it by running `pip install flask` in your terminal.
- Uses decorators to specify different “routes”.
  - For example, the default route is `/`.
  - e.g. `127.0.0.1/`

Flask Basics

- Your app is an instance of the `Flask` object from the `flask` library
  - Say `app` is the name of your `Flask` object in the following.
- `@app.route('/route/path', methods=('m1','m2',...)`
  - Will load what is returned by decorated function in the browser.
  - `methods` = Tuple of valid HTTP request methods for endpoint.
    - `('GET')` is default.
- `@app.teardown_appcontext`
  - Flask will execute decorated function upon the end of a session.
  - Decorated functions must take one positional argument.
    - This is because of under the hood Flask stuff we won't get into.

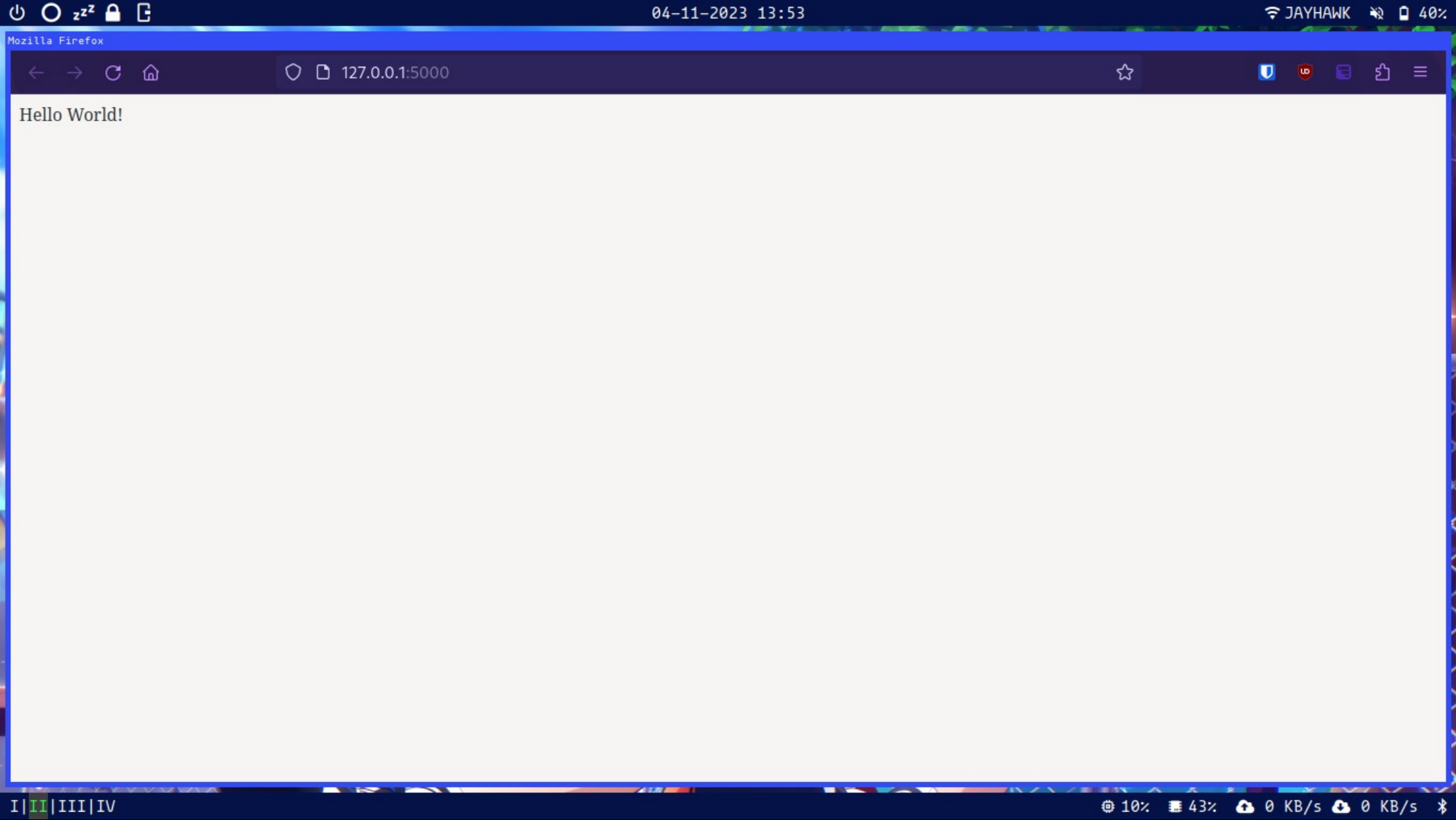
```python
from flask import Flask
app = Flask(__name__)

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

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

app.py file
(Must be named this).

Run app by typing
flask run in the
terminal.

Address of the web app
(type this in your
address bar).

```
[james@localhost ~]$ flask run
* Debug mode: off
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```


04-11-2023 13:53
JAYHAWK
40%
Mozilla Firefox
127.0.0.1:5000
Hello World!
I|II|III|IV
10%
43%
0 KB/s
0 KB/s

Rendering HTML With Flask

- Flask can show an HTML page using `render_template('filename.html')`.
  - File must be in a folder called `templates` located in the same directory as `app.py`.
- However, Flask lets us do more...

HTML Templates With Jinja

- The Jinja template engine is used by Flask to dynamically generate HTML based on an input.
- Pass data to a template in the `render_template`.
  - e.g. `render_template('temp.html', var1=val1, var2=val2, …)` would give us access to all variables passed within the HTML.
- Access variables in template using `{{ varname }}` syntax. You can include any valid Python in this block.
  - e.g. `{{ var1 + var2 }}`

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def hello():
    return render_template('hello.html', name='james')

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

app.py

```
<!DOCTYPE HTML>
<html>
  <body>
    <p>Hello, {{name}}!</p>
  </body>
</html>
```

templates/hello.html

Mozilla Firefox

127.0.0.1:5000

Hello, james!

More Fun with Templates

- **Templates also support:**
  - `if` **statements.**
  - `for` **loops.**
- **Works just like Python.**

```
{% if condition %}
Some html goes here…
{% else %}
Some more html goes here…
{% endif %}
```

```
{% for var in  obj %}
Some html goes here…
{% endfor %}
```

Connecting to a Database

- Connect to a sqlite database using the `sqlite3` library built into Python.
- The `g` object in Flask allows us to add a database connection to our current session.
  - e.g. `g.db = sqlite3.connect('database.db')`
- Make sure to create the database file!

Interacting With a Database

- **After connecting, run** `g.db.execute('sql commnd')` **to interact with your database.**
- `'sql command ?', var` **is used to pass variables to SQL strings.**
  - **Protects from SQL injection attacks.**
  - **e.g.** `db.execute('INSERT INTO names VALUES (?)', (request.form['name'],))`
- **Commands that change the DB will need to be followed by** `g.db.commit()` **to write changes.**
- **If querying, use** `.fetchall()` **to get all results.**
  - **e.g.** `db.execute('SELECT * FROM names').fetchall()`

```
from flask import render_template, Flask, g
import sqlite3

app = Flask(__name__)

def get_db() :
    g.db = sqlite3.connect('database.db')

    g.db.row_factory = sqlite3.Row

    g.db.execute('CREATE TABLE IF NOT EXISTS names(name TEXT)')

    return g.db

@app.teardown_appcontext
def close_db(exception):
    db = g.pop('db', None)

    if db is not None:
        db.close()
```

Processing Requests

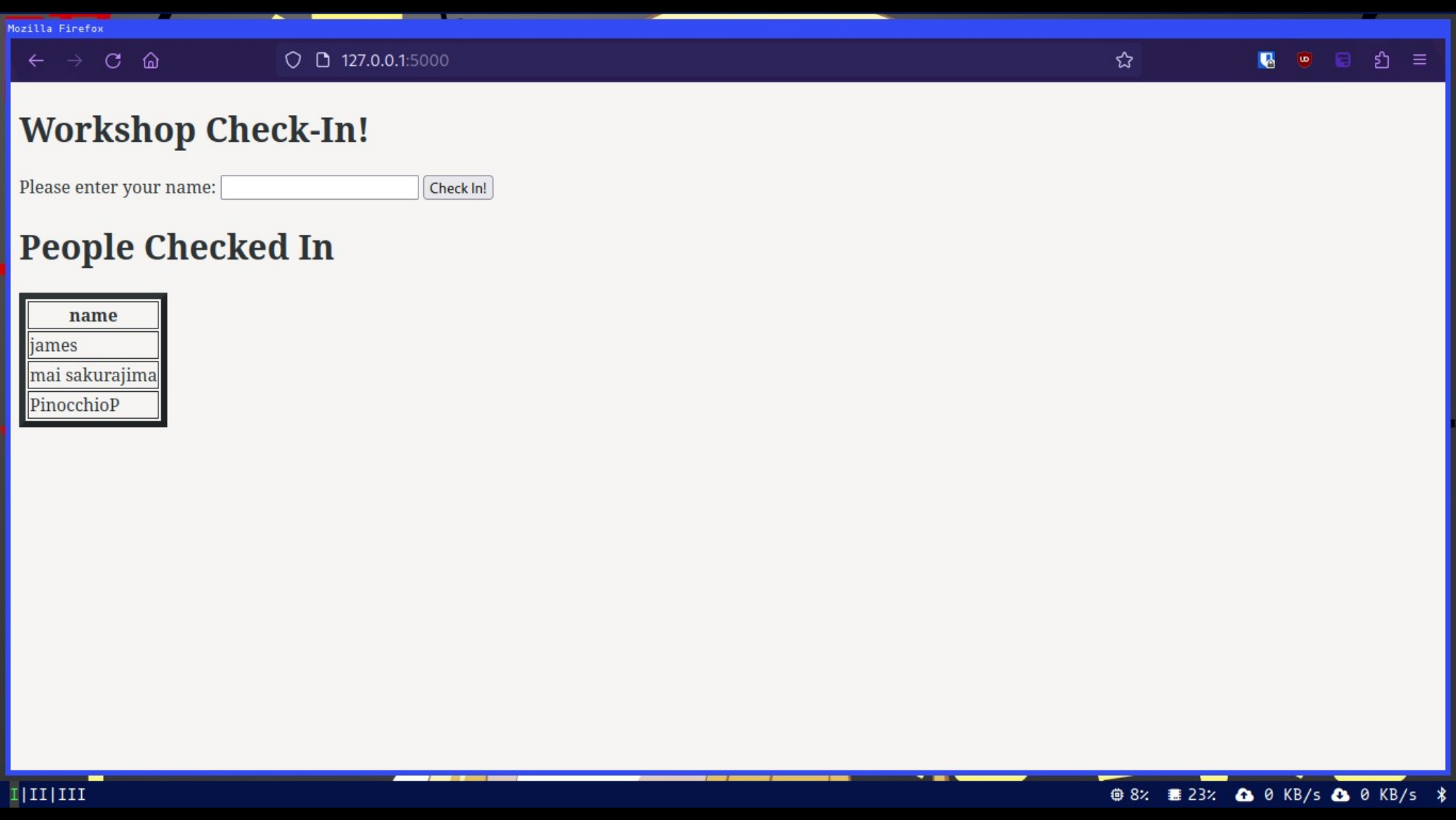
- Import the `request` object from `Flask` to view request data.
- POST form responses will be in the dictionary `request.form`
  - Keys are the names of the `input` tags.
- `request.method` = HTTP method used.

```python
from flask import render_template, request, Flask, g
import sqlite3

app = Flask(__name__)

#database functions from earlier go here

@app.route('/', methods=('GET', 'POST'))
def index():

    db = get_db()

    if request.method == 'POST':
        db.execute('INSERT INTO names VALUES (?)',(request.form['name'],))

        db.commit()

    names = db.execute('SELECT * FROM names').fetchall()

    return render_template('index.html', names=names)

if __name__ == '__main__':
    index()
```

app.py

```
<!DOCTYPE HTML>
<h1>Workshop Check-In!</h1>
  <form method="post">
    <label for="name">Please enter your name: </label>
    <input name="name" required>
    <input type="submit" value="Check In!">
  </form>

<h1>People Checked In</h1>
<table border=6pt>
  <tr>
    <th>name</th>
  </tr>
  {% for name in names %}
  <tr>
    <td>{{name['name']}}</td>
  </tr>
  {% endfor %}
</table>
</section>
```

templates/index.html


Mozilla Firefox
127.0.0.1:5000
Workshop Check-In!
Please enter your name:
Check In!
People Checked In
name
james
mai sakurajima
PinocchioP
I|II|III
8%
23%
0 KB/s
0 KB/s

I am very tired, are we almost done?

One last thing! But first, questions?

git

SECTION:4

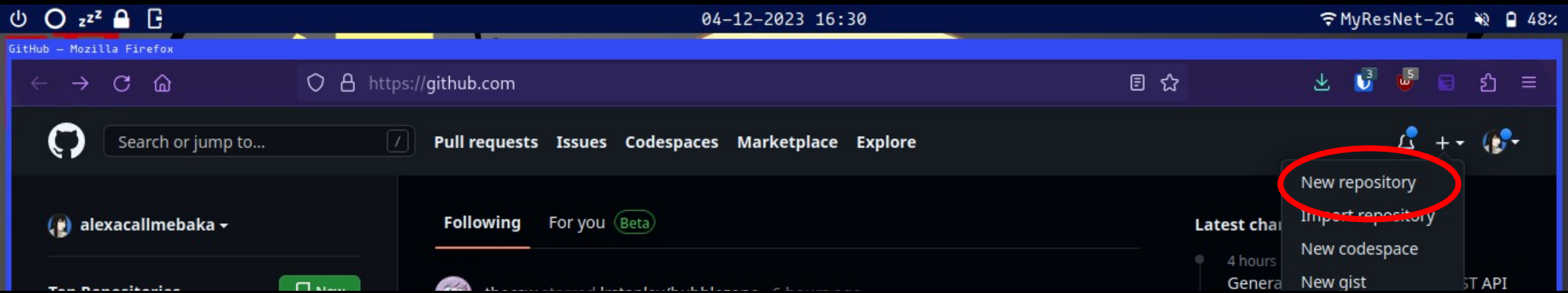
Effective collaboration.

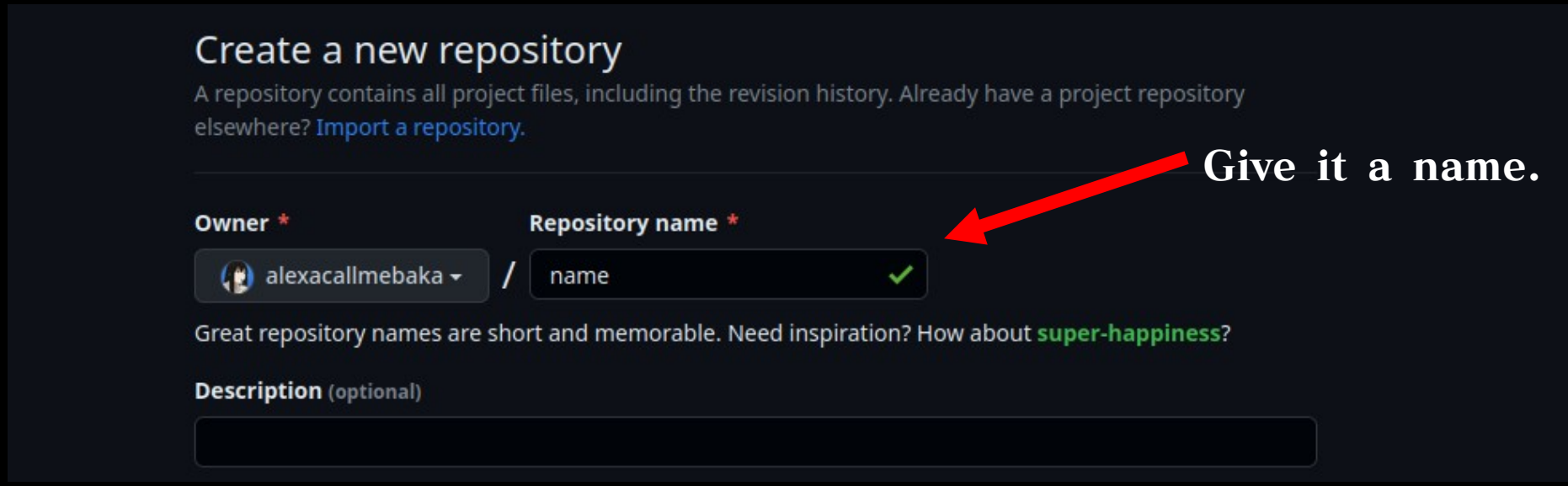
What is git and GitHub?

- git is a version control system (VCS).
  - Saves snapshots of your code at different points called “commits”.
  - Download link on the GitHub for this talk.
- git is useful for many things.
  - We will focus on its use as a collaborative tool.
- GitHub is an online platform for hosting git repositories.
  - This is how you share your code with the world!

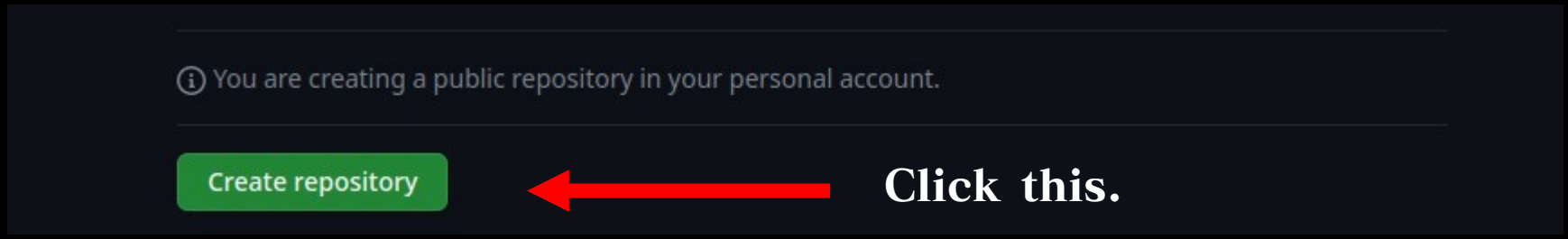
Setting Up GitHub

- Create a GitHub account.
  - `github.com`
- Create a new repository on the home page.



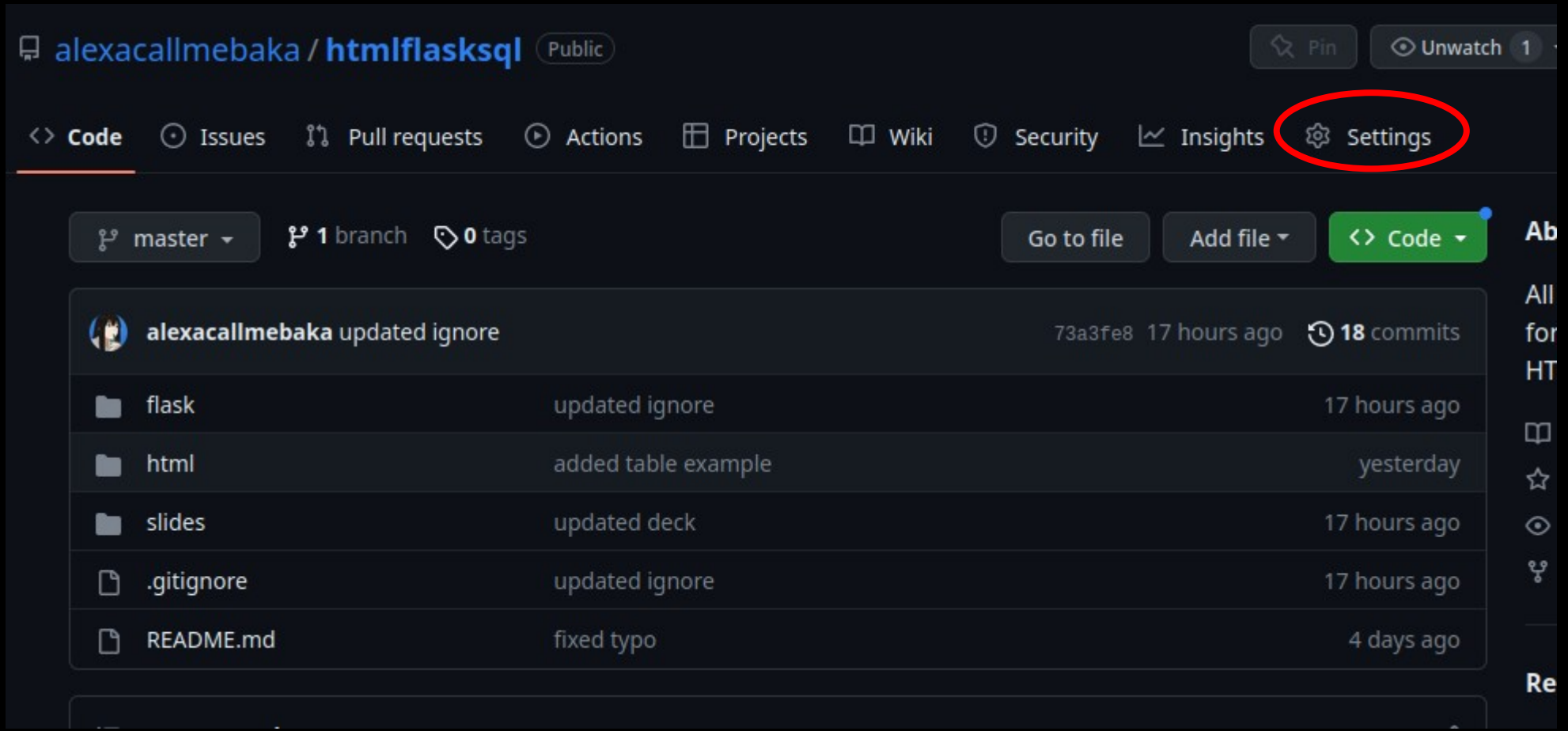

Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.
Give it a name.
Owner *
Repository name *
alexacallmebaka
/
name
Great repository names are short and memorable. Need inspiration? How about super-happiness?
Description (optional)

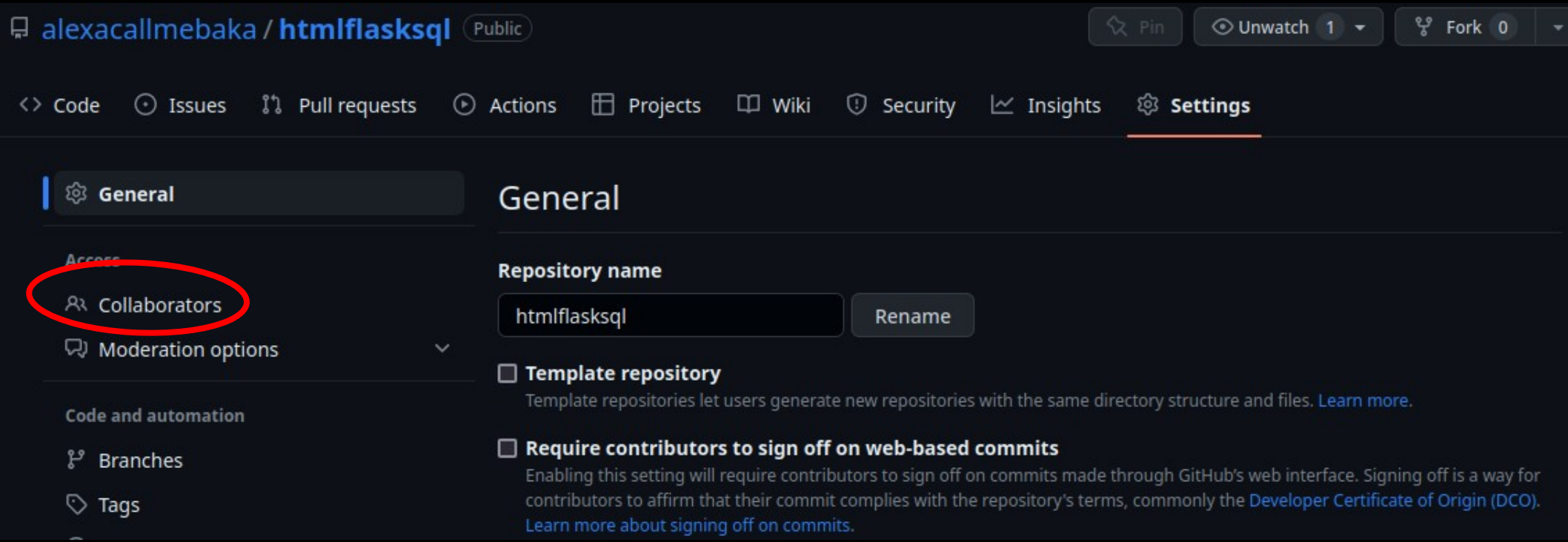
Scroll down…

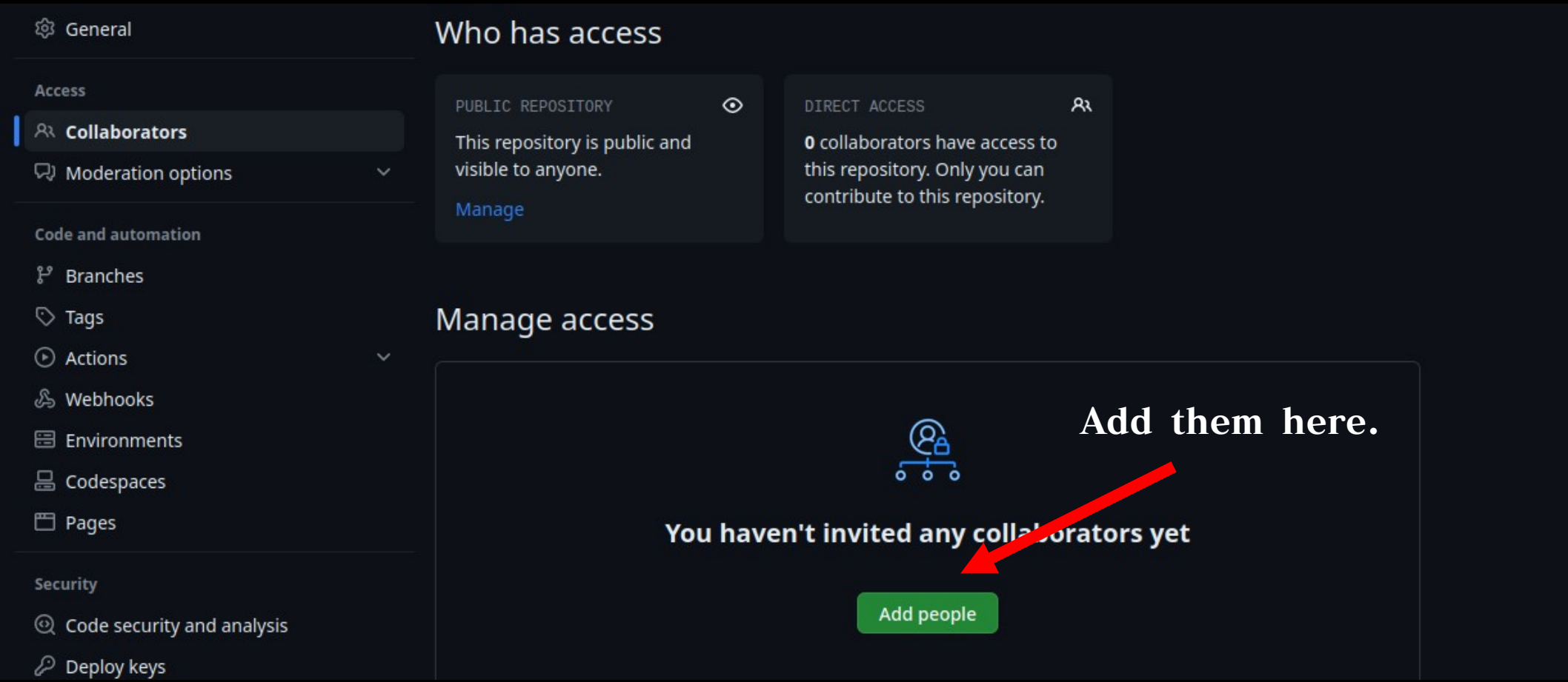

You are creating a public repository in your personal account.
Create repository
Click this.

Whohoo! You made your first GitHub repo!

Now how do you add your friends?


alexacallmebaka / htmlflasksql
Public
Pin
Unwatch 1
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
1 branch
0 tags
Go to file
Add file
Code
alexacallmebaka updated ignore
73a3fe8 17 hours ago
18 commits
flask
updated ignore
17 hours ago
html
added table example
yesterday
slides
updated deck
17 hours ago
.gitignore
updated ignore
17 hours ago
README.md
fixed typo
4 days ago


alexacallmebaka / htmlflasksql
Public
Pin
Unwatch 1
Fork 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
General
Repository name
htmlflasksql
Rename
Template repository
Template repositories let users generate new repositories with the same directory structure and files. Learn more.
Require contributors to sign off on web-based commits
Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO).
Learn more about signing off on commits.


General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Actions
Webhooks
Environments
Codespaces
Pages
Security
Code security and analysis
Deploy keys
Who has access
PUBLIC REPOSITORY
This repository is public and visible to anyone.
Manage
DIRECT ACCESS
0 collaborators have access to this repository. Only you can contribute to this repository.
Manage access
Add them here.
You haven't invited any collaborators yet
Add people

Now we need to actually make a repo to push to GitHub.

Baby's First git Repo

- **Configure git:**
  - `git config --global user.name "james"`
  - `git config --global user.email "jmh@ku.edu"`
  - **NOTE: THESE WILL BE PUBLICLY VISIBLE!!**
- **On the command line, navigate to the directory you want your repo to be in, and run:**
  - `git init`
- **Connect your GitHub repo located at `url`:**
  - `git remote add origin url`
- **Now make some files, go wild!**

Doing Things in git

- `git add filename`
  - **Add `filename` to next commit. Called "staging".**
- `git commit -m 'message'`
  - **Commit all staged files to repo with message `message`.**
- `git push`
  - **Push changes to GitHub.**
  - **May need to run `git push -u origin master` the first time in a new repo.**
- `git clone url`
  - **Copy git repo located at `url` onto your local machine.**
- `git pull`
  - **Pull changes from GitHub.**

That is just a taste of what you can do.

Questions?

Our whirlwind journey has come to an end.

Where to go next?

- This deck, and all example code on GitHub!
- You will also find links to helpful resources.
- Flag me down at HackKU or ping me on the Discord!

hfs.jameshurd.net

Thank you

FIN.

Contact: jameshurd@ku.edu