MLH localhost

Workshop

Basic Training: Intro to Python Skills for AI, Part II

MLH
MAJOR LEAGUE HACKING

Our Mission *is to Empower Hackers.*

65,000+
HACKERS

12,000+
PROJECTS CREATED

400+
CITIES

We hope you learn something awesome today!
Find more resources: http://mlh.io/

Our unique expression of social good

Capital One is dedicated to providing opportunities and resources that will enable more people to succeed.

Through our Future Edge program, we're investing in and collaborating with leading educational and community organizations across the U.S.—to help more people succeed in the 21st century.

We're empowering families through financial literacy and affordable housing, helping individuals bridge the digital skills gap and showing small businesses how to harness technology to grow and compete.

What will you learn today?

1 How Python can be used to create web apps

2 Several useful Python libraries

3 How to create a chat experience in a web app

Why does this matter?

1 Many artificial intelligence projects are on the web. You want to be able to share your cool work with others!

2 Learning how to use third party libraries is a skill you'll use in any programming language.

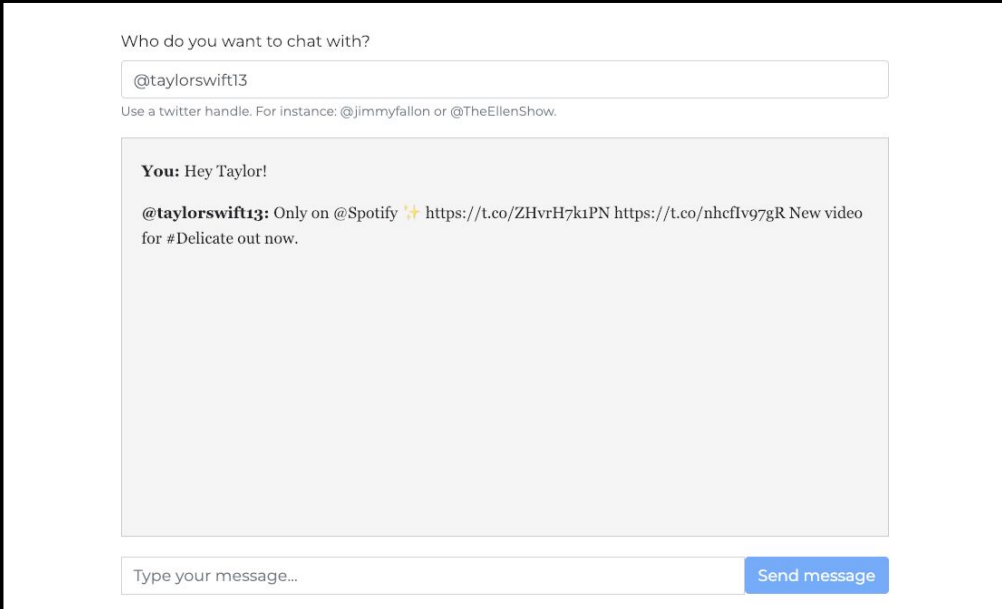**3** Once you learn one programing language, you can learn any .

What do you remember from last time?

Discuss for a few minutes with someone around you.

What are you going to build today?

You're going to use your new Python skills to create a Chat Bot!
Give it a try by heading to the URL below:

https://mlhlocal.host/glitch-bot

How does this work?

1. In the first box (called a text field), the user enters a person's Twitter name.

2. In the second box, the user enters a message.

3. The Twitter name and message are sent to the app.

Who do you want to chat with?

@jimmyfallon

Use a twitter handle. For instance: @jimmyfallon or @TheEllenShow.

Type your message...   Send message

How does this work?

- The app uses a library called **BeautifulSoup** to scrape up to 10,000 Tweets from a user using the Twitter API

- The app uses a Python library called **Flask** to handle the requests and responses.

- Then, the app randomly returns one of the Tweets to you!

What's web scraping?

Lots of information on the internet is text. Python has some libraries that you can use to easily **scrape** that text, or copy it from the website into a file.

Most libraries have documentation to help you learn how to use the library.

| Key Term |
| --- |
| **Scrape:** To collect all of the data (code) from a webpage |

What's web scraping?

You can see the Beautiful Soup documentation here:

`mlhlocal.host/beautifulsoup`

You won't need the documentation today, but you should get used to reading and learning from documentation. It's a very important skill for all developers to have!

How are you going to rebuild it?

These are the steps you'll need to take:

1. You'll make your own copy of the code on Glitch.

2. You'll read through the code and discuss it with others.

3. You'll write some missing Python code.

4. Then, you'll recap and take a quiz!
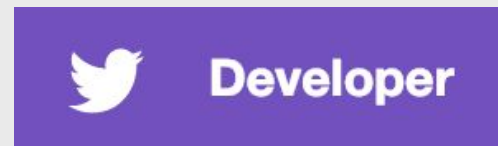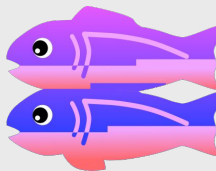
That may sound like a lot of steps. Part of coding/programming is learning how all the different tools work and how they work together.

We know you can do it!

Some things to keep in mind

Writing code is tricky. Pay attention to the following best practices!

- The majority of the time, when something isn't working with your code, it's as simple as a typo. Make sure that you've typed everything correctly before moving on to other debugging techniques.

- Glitch has a really cool rewind feature. If your code was working before, and it's not working now, use the rewind feature to back up.

- It's okay to make mistakes! You're learning something new, and that's really tricky.
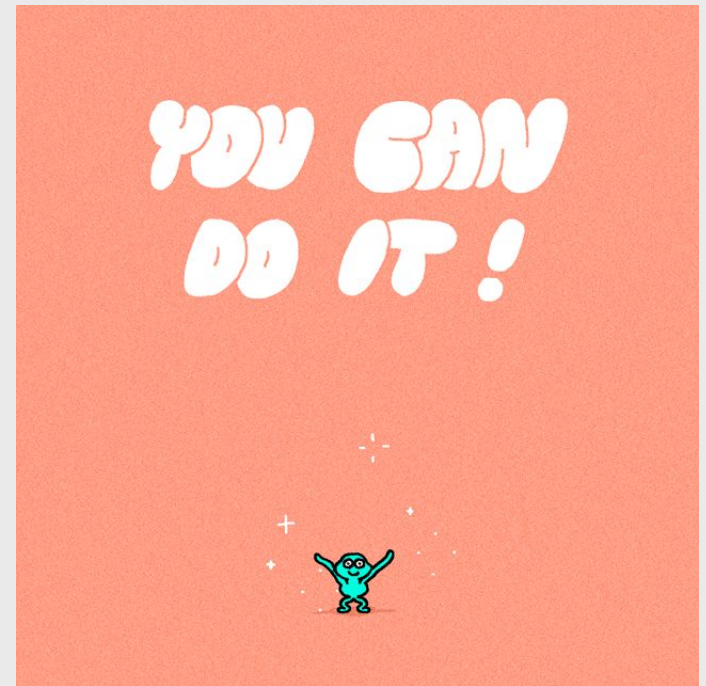
Let's get started!

Table of Contents

What Python Skills Did We Learn Last Time?

1. **Variables:** variables in programming are a lot like variables in math! They store information for us.

2. **Loops:** loops are a structure your program can use to perform the same task multiple times (rather than coding it out by hand every time).

3. **Functions:** a structure you can use in your program to reuse bits of code throughout the program

4. **Libraries:** a collection of functions someone wrote, which we can import into our own programs to use.

17
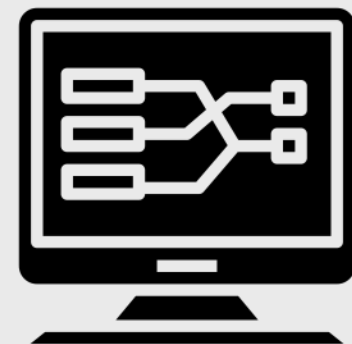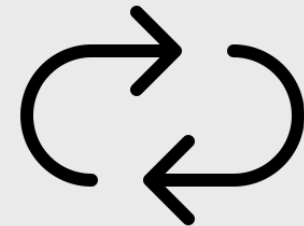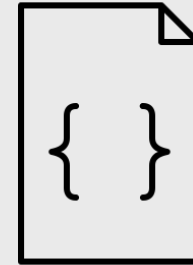
What Will You Learn Today?

You will learn the following Python skills so that you can clean some data.

1. **Regular expressions** – a programming concept used to work with text

2. **Imports** – using data or functions from other sources (like libraries or other files in our project)

Key Term
Data cleaning: using code to remove or edit parts of data that we don't want, to standardize it and make it easier to work with

The best way to learn to code is to CODE, not to read about it, so let's dive in!

Table of Contents

Glitch

Please note! Signing in is optional, but it lets you save your code.

Instructions are below.

1. Navigate to this URL.

glitch.com

2. Click **Sign in**. Choose how you want to log in.

Get the code

https://mlhlocal.host/glitch-python-ii
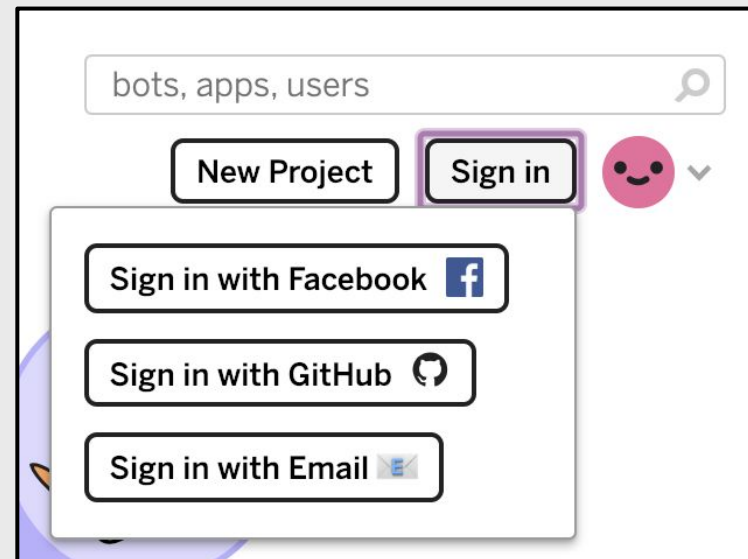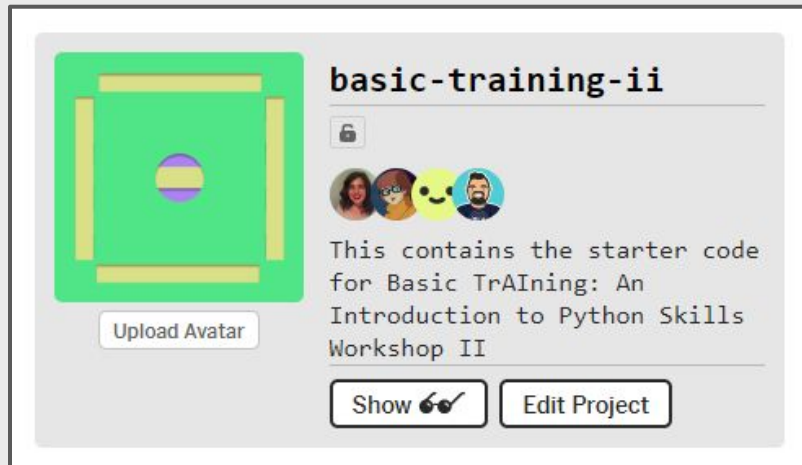
- Navigate to the URL above.

- Click **Edit Project**.

Get the code
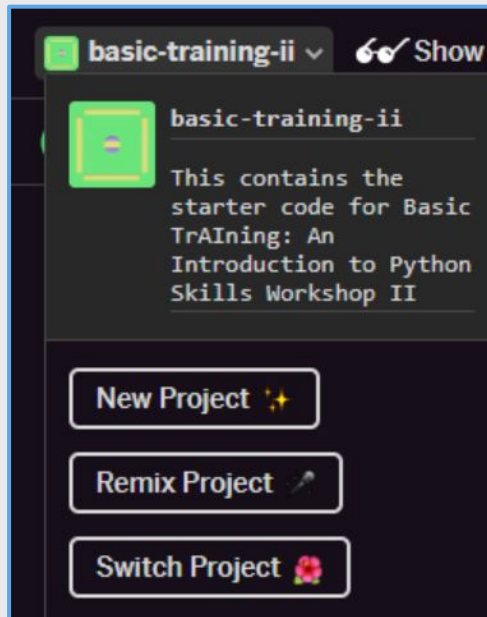
- Click the **basic-training-ii** title on the top left of the screen

- Then click **Remix Project** to create a copy of the code.

Now you have all the code you need to complete this workshop! Let's explore it.

What files are in this project?

```
回 assets
> static/
> templates/
𝒪⚷ .env
.gitignore
LICENSE
README.md
config.py
glitch.json
main.py
markov_bot.py
requirements.txt
resources.txt
twitter_scraper_f… r.py        ⋮
```

logo.svg is the logo image

script.js is a JavaScript file that adds the bot responses to the center box on the webpage

style.css is the file that styles our webpage

homepage.html is the file that provides the structure of our webpage, such as the text fields where the user types their question

config.py is a file that tells Glitch how to run our app

What files are in this project?

```
🗄 assets
> static/
> templates/
🗝 .env
.gitignore
LICENSE
README.md
config.py
glitch.json
main.py
markov_bot.py
requirements.txt
resources.txt
twitter_scraper_f… r.py        ⋮
```

main.py is a Python file that handles the requests and responses for our file. When a user types a message into the chat box, they are requesting the bot to send a message back.

twitter_scraper_fetcher.py is a file that scrapes Tweets. We'll be working with this one today!

README.md is a file that tells you about the project. The words inside of it are not code.

requirements.txt is a file that tells our project what libraries and frameworks that it needs to work.

resources.md contains all sorts of resources to keep learning later!

Table of Contents

Now that you understand the project, dive in!

Running the Glitch App
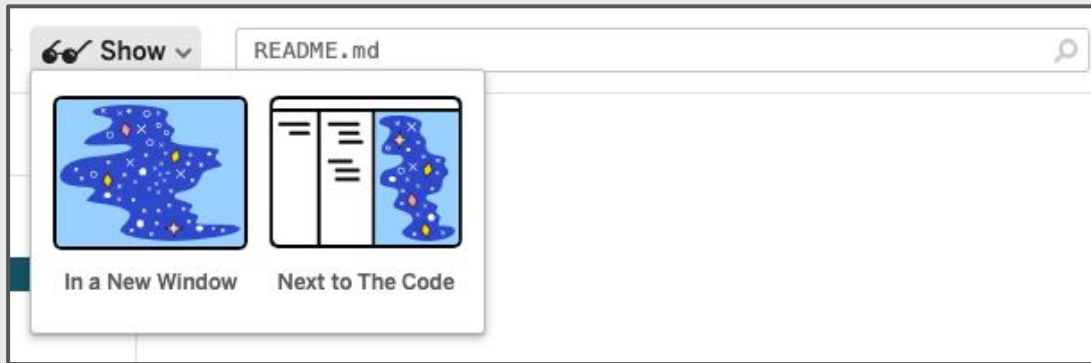
You can run the app by clicking **Show** in the top bar.

But we aren't ready to run the code- we have to write some functions first!

Let's get started!
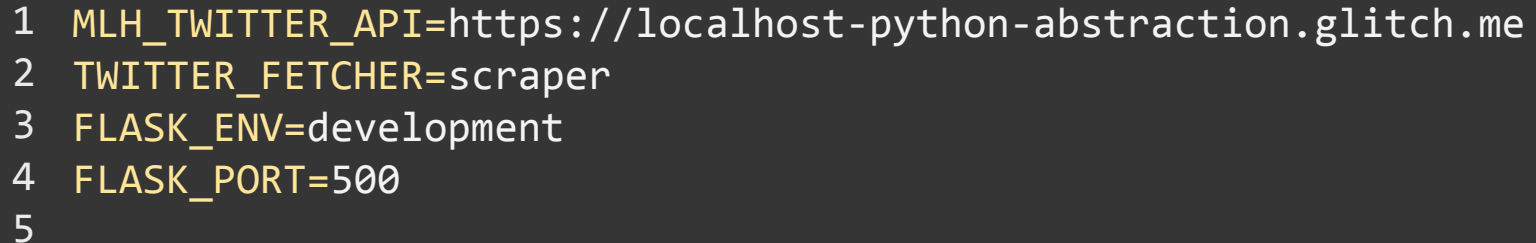
.env Credentials

The .env file contains the Environment Credentials we need to run the application.

Open up the **.env** file and add the following details so that it looks like this:

```
1  MLH_TWITTER_API=https://localhost-python-abstraction.glitch.me
2  TWITTER_FETCHER=scraper
3  FLASK_ENV=development
4  FLASK_PORT=500
5
```

Let's write some code!

Open **`twitter_scraper_fetcher.py`**.

Notice that we have three empty functions - this is what where you are going to write some code.

`get_elements()` returns all the data from a Twitter account.

`get_user_tweets()` extracts the tweet data from the elements data.

`clean_tweets_data()` removes emojis, URLs, and other data we don't want from the tweets.

```python
11  def get_elements(twitter_handle):
12      ## Nothing here yet!
13      pass
14
15  def get_user_tweets(twitter_handle):
16      ## Nothing here yet!
17      pass
18
19  def clean_tweets_data(tweets):
20      ## Nothing here yet!
21      pass
22
```

Write Code: `get_elements()`

We're going to use libraries called **Requests** and **BeautifulSoup** which can scrape data from websites for us.

```
14 def get_elements(twitter_handle):
15     url = TWITTER_URL + twitter_handle
16
```

This function has one parameter, **twitter_handle**, which represents a twitter username.

Line 12: We create a full twitter URL.
This works by adding two Strings together: The Twitter URL on **Line 8**, and a twitter handle passed into the function as a parameter.

For example, TWITTER_URL + 'nasa' would return **http://twitter.com/nasa**.

Write Code: `get_elements()`

We're going to use libraries called **Requests** and **BeautifulSoup** which can scrape data from websites for us.

```
14 def get_elements(twitter_handle):
15     url = TWITTER_URL + twitter_handle
16     response = requests.get(url, headers={'User-Agent': AGENTS})
17     html = response.content
18
```

Line 13: We use the **Requests** library to return all the HTML (data) from that webpage - the URL we created on Line 12.

The **User-Agent** field is a string that tells Twitter which device and browser we are using.

Line 14: From the response we receive from the webpage, we just want the content.

33

Write Code: `get_elements()`

Add a few more lines to the function.

```python
14 def get_elements(twitter_handle):
15     url = TWITTER_URL + twitter_handle
16     response = requests.get(url, headers={'User-Agent': AGENTS})
17     html = response.content
18
19     soup = BeautifulSoup(html, "html.parser")
20
```

Line 16: We create a variable called **soup** which is a new **BeautifulSoup** object. We create it and say that we want it to analyse the HTML content we created on **Line 14**.

Write Code: `get_elements()`

Add a few more lines to the function.

```python
def get_elements(twitter_handle):
    url = TWITTER_URL + twitter_handle
    response = requests.get(url, headers={'User-Agent': AGENTS})
    html = response.content

    soup = BeautifulSoup(html, "html.parser")

    return soup.find_all(CONTENT_CONTAINER_TAGS, {"class": CONTENT_CLASS_NAME})


```

Line 18: We put our **soup** variable to use and ask it to find all of the relevant information we need, and return it.

Write Code: `get_elements()`

What's happening?

Line 16: We create a `BeautifulSoup` object. The `BeautifulSoup` library makes it easy to parse large amounts of data.

Lines 18-20: We use the object to search through the HTML and give us all the data that contains tweets.

Key Term
Parse: The process of analysing large sets of data with algorithms.

Write code: `get_user_tweets()`

Let's move onto the next function: **get_user_tweets()**. This will return a list of a Twitter user's recent tweets.

Add the following code to the function:

```
23  def get_user_tweets(twitter_handle):
24     elements = get_elements(twitter_handle)
25
26
```

Line 20: Our function has one parameter - the user's twitter handle.

Line 21: We call the function we wrote above, **get_elements()**!

Write code: `get_user_tweets()`

```
23 def get_user_tweets(twitter_handle):
24    elements = get_elements(twitter_handle)
25    tweets = []
26
```

Line 22: We create an empty list called `tweets` - this is going to contain a list of tweets we extract using **BeautifulSoup**.

Write code: `get_user_tweets()`

Add a few more lines to the **get_user_tweets()** function:

```
23  def get_user_tweets(twitter_handle):
24    elements = get_elements(twitter_handle)
25    tweets = []
26
27    for post in elements:
28        for text in post.contents:
29
```

Lines 24-25: We loop over each item in the **elements** list.

Then for each of *those* items, we loop over all of the contents.

Write code: `get_user_tweets()`

```python
23  def get_user_tweets(twitter_handle):
24     elements = get_elements(twitter_handle)
25     tweets = []
26
27     for post in elements:
28         for text in post.contents:
29             # check if line contains real text
30             if text.string not in EMPTY_ITEMS:
31                 tweets.append(text.string)
32     return tweets
```

Line 27: If the text contains some data (is **not** empty), add it to our list of Tweets.

Line 29: Return the list of Tweets.

Review Code: `get_user_tweets()`

What's happening?

Line 26-30: We have a loop inside a loop!

What we are doing is looping through all of the HTML elements, and checking the text to see if they contain information.

If they do (`if not in EMPTY_ITEMS`), we append (add) it to our list of tweets.

Line 31: We return the list of tweets that we have created.

Awesome!

Now, we've got one more function to add...

Regular Expressions

We're going to clean up our tweets with a powerful programming tool called **Regular Expressions**.

A Regular Expression (or regex) is a sequence of characters which create a search pattern.

We can use it to remove all data that doesn't match our regex search criteria.

How do they work?

Regular Expressions: Example

First we create a regex pattern:

```
url_pattern = re.compile(r"http\S+", re.DOTALL)
```

- `re` is the Python regex library.
- **`r”http\S+”`** means: create a regular expression (**`r`**) that finds text that starts with http and is followed by any length of string (**`\S+`**).
- **`re.DOTALL`** means apply this to everything, including new lines.

Regular Expressions: Example

Then, we can use it to replace and remove text that contains that search pattern.

```
example_tweet = "check out this URL! https://myamazingURL.com"
text_without_url = url_pattern.sub(r"", example_tweet)
```

We use the **sub** (substitute) function to replace any text that matches our search pattern with nothing (**""** - an empty string).

The variable **text_without_url** would then equal:

"check out this URL!"

Regular Expressions: Example

Regular Expressions are tricky!

We won't ask you to write some search patterns from scratch - we'll let you know which ones to use.

Our regular expressions will remove URLs, emojis, and other data we don't want to display.

Write Code: `twitter_scraper_fetcher.py`

Open **resources.txt:**

```
1   ## Use in twitter_scraper_fetcher.py
2
3   emoji_pattern = re.compile(
4           "["
5           u"\U0001F600-\U0001F64F"  # emoticons
6           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
7           u"\U0001F680-\U0001F6FF"  # transport & map symbols
8           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
9           "]+",
10          flags=re.UNICODE,
```

4.   Copy lines 3 through 11.

These are some regular expressions which will search for ranges of Unicode characters that are emojis.

For example, the Unicode character **U0001F600** is the code representation of the grinning emoji!

Write Code: `clean_tweets_data()`

We're going to add this to our final function, **clean_tweets_data()**.

Copy the regex and add it to the function:

```
34   def clean_tweets_data(tweets):
35     emoji_pattern = re.compile(
36         "["
37         u"\U0001F600-\U0001F64F"   # emoticons
38         u"\U0001F300-\U0001F5FF"   # symbols & pictographs
39         u"\U0001F680-\U0001F6FF"   # transport & map symbols
40         u"\U0001F1E0-\U0001F1FF"   # flags (iOS)
41         "]+",
42         flags=re.UNICODE,
43     )
```

Write Code: `clean_tweets_data()`

Now, we're going to add a few more regex to remove URLs and Tweets that mention other users:

```python
44    url_pattern = re.compile(r"http\S+", re.DOTALL)
45    mentions_pattern = re.compile(r"@\S+", re.DOTALL)
46
47    cleaned_tweets = []
48    for tweet in tweets:
49      text_without_emoji = emoji_pattern.sub(r"", tweet)
50      text_without_url = url_pattern.sub(r"", text_without_emoji)
51      cleaned_tweets.append(mentions_pattern.sub(r"", text_without_url))
52
53    return cleaned_tweets
54
```

Lines 43-46: We apply all of our regex to each tweet in our list.

Line 48: Returns our newly cleaned tweets!

Review: `clean_tweets_data()`

```python
34  def clean_tweets_data(tweets):
35    emoji_pattern = re.compile(
36          "["
37          u"\U0001F600-\U0001F64F"  # emoticons
38          u"\U0001F300-\U0001F5FF"  # symbols & pictographs
39          u"\U0001F680-\U0001F6FF"  # transport & map symbols
40          u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
41          "]+",
42          flags=re.UNICODE,
43      )
44    url_pattern = re.compile(r"http\S+", re.DOTALL)
45    mentions_pattern = re.compile(r"@\S+", re.DOTALL)
46
47    cleaned_tweets = []
48    for tweet in tweets:
49      text_without_emoji = emoji_pattern.sub(r"", tweet)
50      text_without_url = url_pattern.sub(r"", text_without_emoji)
51      cleaned_tweets.append(mentions_pattern.sub(r"", text_without_url))
52
53    return cleaned_tweets
```

What did you just do?

You created a new function that will remove all sorts of unnecessary characters from our Tweets.

Code Review: `main.py`

Let's take a look at the rest of the main.py and see how it returns a tweet to the user.

```python
17  # Chat API - WebSocket
18  @socketio.on("send question")
19  def generate_message(body, methods=["POST"]):
20      question = body["message"]
21      twitter_handle = body["username"]
```

Lines 17-20: When a user enters a question on our web page, save their message in a variable called **question** and save the **twitter_handle** of the person they want to talk to.

Code Review: `main.py`

```
23    try:
24        # Send the answer to the app, to display to the user
25        bot_answer = generate_bot_answer(twitter_handle, question)
26        answer = {"username": twitter_handle, "message": bot_answer}
27        socketio.emit("bot answer", answer)
28
```

Lines 23-27: This is called a **try** block. The program will first attempt to do the code inside of this block, and if it doesn't work it will go to the **except** block below.

Line 26: Save the **twitter_handle** and **bot_answer** to a dictionary called **answer**.

Line 28: Send the answer to the user! That's how we get this:

Code Review: `main.py`

Who do you want to chat with?

@Nasa

Use a twitter handle. For instance: @jimmyfallon or @TheEllenShow.

You: Hey, Nasa!

@Nasa: and Boeing teams are working overtime to accelerate the launch schedule of

answer

twitter_handle   bot_answer

```
23   try:
24       # Send the answer to the app, to display to the user
25       bot_answer = generate_bot_answer(twitter_handle, question)
26       answer = {"username": twitter_handle, "message": bot_answer}
27       socketio.emit("bot answer", answer)
28
```

Code Review: `main.py`

```
26    except:
27        bot_answer = "Sorry, I couldn't process that. Try again please."
28        socketio.emit("error", {"username": twitter_handle, "message":
29        bot_answer})
30
```

Lines 26-30: This is called an **except** block. If the code in the **try** block doesn't work, the program will send this error message to the user. You might remember this error message from the first time you ran the app, right?

Try the app!

Click **Show** and try your app now!

Who do you want to chat with?

@Nasa

Use a twitter handle. For instance: @jimmyfallon or @TheEllenShow.

You: Hey!

@Nasa: asked "After the Soyuz docked successfully why does it take so long for the hatch opening?"

What changed?

Now, your bot returns something a lot more like a sentence, rather than random emojis and @mentions.

What next?

In the next workshop, we're going to incorporate artificial intelligence (AI) so that our app actually responds to our message, rather than just returning a random Tweet!

Who do you want to chat with?

@neiltyson

Use a twitter handle. For instance: @jimmyfallon or @TheEllenShow.

You: Hey Neil! Tell me something cool.

@neiltyson: Congratulations to Jimmy Carter for Best Spoken Word at the

Table of Contents

1. Recap

2. Explore the code

3. Write Code!

4. Review & Quiz

5. Next Steps

Let's Recap

What did you learn:

1 Python is a programming language you can use for many things.

2 Regular expressions are used to clean data.

3 Python tools you can use include imports, returns, and try/except blocks.

What did you learn today?

We created a fun quiz to test your knowledge and see what you learned from this workshop.

http://mlhlocal.host/quiz

Table of Contents

1. Recap

2. Explore the code

3. Write Code!

4. Review & Quiz

5. Next Steps

Next Steps

Where to go from here...

1 There is a lot of code in this app that we didn't get to review. Try to read it and understand it on your own!

2 Check out codecademy.com's free Python lessons.

3 Take the third workshop!