Learning Objectives

Learners will be able to...

- **Utilize ChatGPT to generate code snippets for specific programming tasks or challenges.**
- **Integrate ChatGPT-generated code into both front-end and back-end development projects.**
- **Generate code in programming languages that are unfamiliar to the learner using ChatGPT.**
- **Evaluate the quality and accuracy of ChatGPT-generated code and refine it as needed.**

info

Make Sure You Know

You are familiar with Python.

Limitations

This is a gentle introduction. So there is a little bit of Python programming. The information might not be the most up to data as OpenAI releases new features.

Flask

In this course, we’ll be exploring the fun world of natural language processing and learning how to use the ChatGPT API to build a website that can generate responses to user input. By integrating the ChatGPT API into a web application, we can create a responsive and interactive website that can engage users and provide them with valuable insights and information.

we’ll be using the Flask web framework and the OpenAI Python library to build a website that can generate responses to user input using the ChatGPT API. We’ll start by setting up our development environment and creating a basic user interface, and then move on to integrating the ChatGPT API and optimizing website performance and user experience.

By the end of this course, you’ll have a solid understanding of how to use the ChatGPT API to build a website, and you’ll be equipped with the skills and knowledge to start building your own natural language processing applications. Let’s get started!

Setting up

Before we can start building our website with the ChatGPT API, we'll need to set up our development environment. Here's what we'll need:

Python: We'll be using Python 3.7 or later to build our web application. you can run the following code in terminal to check the python version:

```
python -V
```

Flask: Flask is a lightweight web framework for Python that will allow us to easily build our web application.Once you have Python installed, you can install Flask using pip, Python's package manager. from your terminal window run the following command:

```
pip install flask
```

OpenAI API key: To use the ChatGPT API, we'll need an API key from OpenAI. Which we have been using the entire course.

Python libraries: We'll be using several Python libraries, including the OpenAI Python library and the dotenv library for managing environment variables.

Once we have all the necessary tools and libraries installed, we can start building our web application. Here's how to set up our development environment:

Install Python libraries: We'll also need to install several Python libraries using pip. The following libraries wont be necessary while inside Codio but will mention them if want to end up working outside the plaform. In addition to Flask and the OpenAI Python library, you will need to use the following library for managing environment variables if outside the codio platform. Run the following commands to install these libraries:

```
pip install openai
pip install python-dotenv
```

With all these tools in place we can start building our application.

Using ChatGPT To create our html

You will be given two files an html file and a python File. we will use the python file to use GPT API Call to generate the html file. Please not the response that we get can differ because of that we will also show the code

```
import openai

# Replace 'your_api_key' with your actual API key
openai.api_key = "your_api_key"

def generate_website():
  prompt = " In this HTML file, we define a basic structure for
        our website that includes a title, a header, a form for
        user input, and a script tag that loads our JavaScript
        code. We also include a link to a CSS file (style.css)
        that defines the styling for our website."
  response=openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a html expert you
        are going to write a html file"},
        {"role": "user", "content": prompt}
    ]
)
  website_code = response['choices'][0]['message']
        ['content'].strip()
  return website_code
```

Ask chatGPT to create the html and css code.

```
You are a html expert you are going to write a html file. In
this HTML file, we define a basic structure for our website that
includes a title, a header, a form for user input, and a script
tag that loads our JavaScript code. We also include a link to a
CSS file (style.css) that defines the styling for our website.
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>ChatGPT API Demo</title>
    <link rel="stylesheet" href="static/style.css">
  </head>
  <body>
    <h1>ChatGPT API Demo</h1>
    <form id="chat-form">
      <div class="input-group">
        <label for="user-input">User Input:</label>
        <input type="text" id="user-input" name="user-input">
      </div>
      <div class="input-group">
        <label for="response">Response:</label>
        <textarea id="response" name="response" readonly>
        </textarea>
      </div>
      <button type="submit" id="submit-button">Submit</button>
    </form>
    <script src="static/app.js"></script>
  </body>
</html>
```

Creating our CSS

Now we are going to change our prompt to create the css.

In order to generate the css, we are going to change our prompt, had make it clear we just want the css.

```
prompt = " In this HTML file, we define a basic structure for
our
 website that includes a title, a header, a form for user input,
and
  a script tag that loads our JavaScript code. We also include a
link
   to a CSS file (style.css) that defines the styling for our
website
```

```css
body {
  font-family: Arial, sans-serif;
}

h1 {
  text-align: center;
}

form {
  max-width: 800px;
  margin: 0 auto;
}

.input-group {
  display: flex;
  flex-direction: column;
  margin-bottom: 10px;
}

label {
  font-weight: bold;
}

textarea {
  height: 100px;
}

#submit-button {
  margin-top: 20px;
}
```

Javascript

we’ll create our JavaScript code that handles user input and sends requests to the ChatGPT API.

Now we are going to modify our prompt so that it can generate for us the javascript file. insted of telling it to write the css part we will add

```
create our JavaScript code that handles user input and sends
requests to the ChatGPT API.
```

Here’s the code for our app.js file:

```
const form = document.getElementById("chat-form");
const userInput = document.getElementById("user-input");
const responseElem = document.getElementById("response");

form.addEventListener("submit", async (event) => {
  event.preventDefault();

  const userInputValue = userInput.value.trim();

  if (!userInputValue) {
    return;
  }

  const response = await fetch("/chat", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ user_input: userInputValue }),
  });

  const responseData = await response.json();

  if (response.ok) {
    const chatResponse = responseData.response;
    responseElem.value = chatResponse;
  } else {
    alert("Error fetching response from ChatGPT API.");
  }
});
```

Building the Server-Side Code

Now that we have our user interface in place, we need to build the server-side code that will handle user requests and send requests to the ChatGPT API. In this section, we’ll use Flask to create a simple server that receives user input and sends requests to the ChatGPT API.

Here’s the code for our Flask server:

```
import openai
from flask import Flask, jsonify, request, render_template
import os

# Set environment variables
keys = os.getenv('OPENAI_KEY')


api_key= keys

app = Flask(__name__)
openai.api_key = api_key

def generate_chat_response(user_input):
  prompt = user_input
  response=openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are an ai
assistant."},
        {"role": "user", "content": prompt}]
)
  chat_response = response['choices'][0]['message']
['content'].strip()
  return chat_response

@app.route("/chat", methods=["POST"])
def chat():
    user_input = request.json["user_input"]
    chat_response = generate_chat_response(user_input)
    response_data = {"response": chat_response}
    return jsonify(response_data)

@app.route("/")
@app.route("/index.html")
def index():
   return render_template("index.html")

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

In this Flask server code, we define a function generate_chat_response that takes a user input and sends a request to the ChatGPT API to generate a response. We then define a Flask route /chat that handles POST requests with the user input in the request body. The route function calls the generate_chat_response function and returns the response in JSON format.

To run the Flask server, we can add the following lines of code to the end of our Python file:

```
if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

This will start the Flask server in debug mode and allow us to test our application.

Now that we have both our user interface and server-side code in place, we can run our web application and test it out. To start the application, we can run the following command in our terminal:

```
python3 app.py
```

This will start the Flask server, and we can navigate to http://localhost:5000 in our web browser to view the website. When we enter text in the user input field and submit the form, our server-side code will send a request to the ChatGPT API and display the response in the response text area.