

Exploring GPT-3

An unofficial first look at the general-purpose language processing API from OpenAI



Steve Tingiris

Foreword by Bret Kinsella (Founder and CEO of Voicebot.ai)



Preface

What if this audiobook was written by artificial intelligence? Would you read it? I hope so because parts of it were. Yes, GPT-3 was used to create parts of this audiobook. It's a bit meta I know, a audiobook about GPT-3 written by GPT-3. But creating content is one of the many great uses for GPT-3. So why not? Also, for me, content generation was the use case that most piqued my interest. I wondered if GPT-3 could be used in a product I was working on to automate the generation of technical learning material.

You probably also have a specific reason why you're interested in GPT-3. Perhaps it's intellectual curiosity. Or maybe you have an idea that you think GPT-3 can enable. You've likely seen online demos of GPT-3 generating content, writing code, penning poetry, or something else, and you're wondering if GPT-3 could be used for an idea you have. If so, this audiobook was written specifically for you.

My goal for this audiobook is to provide a practical resource to help you get started with GPT-3, as quickly as possible, without any required technical background. That said, as I write this, GPT-3 is still in private beta. So, everyone is learning as they go. But the one thing I've learned for sure is that the possible applications for GPT-3 are vast and there is no way to know all of what's possible, let alone get it into a audiobook. So, I hope this audiobook makes getting started easy, but I also hope it's just the beginning of your journey *Exploring GPT-3*

Who this audiobook is for

This audiobook is for anyone with an interest in NLP or learning GPT-3 — with or without a technical background. Developers, product managers, entrepreneurs, and hobbyists who want to learn about NLP, AI, and GPT-3 will find this audiobook useful. Basic computer skills are all you need to get the most out of the audiobook. While experience with a modern programming language is helpful, it's not required. The code examples provided are beginner friendly and easy to follow, even if you're brand new to writing code.

What this audiobook covers

Chapter 1, Introducing GPT-3 and the OpenAI API, is a high-level introduction to GPT-3 and the OpenAI API.

Chapter 2, GPT-3 Applications and Use Cases, is an overview of core GPT-3 use cases: text generation, classification, and semantic search.

Chapter 3, Working with the OpenAI Playground, is a semi-deep dive into the OpenAI Playground and the developer portal.

Chapter 4, Working with the OpenAI API, is an introduction to calling the OpenAI API using Postman.

Chapter 5, Calling the OpenAI API in Code, is an introduction to using the OpenAI API with both Node.js/JavaScript and Python.

Chapter 6, Content Filtering, explains how to implement content filtering.

Chapter 7, Generating and Transforming Text, contains code and prompt examples for generating and transforming text.

Chapter 8, Classifying and Categorizing Text, takes a closer look at text classification and the OpenAI API Classification endpoint.

Chapter 9, Building a GPT-3 Powered Question-Answering App, explains how to build a functional GPT-3 powered web knowledge base.

Chapter 10, Going Live with OpenAI-Powered Apps, explains the OpenAI application review and approval process and discusses getting ready for a review.

To get the most out of this audiobook

All of the code examples in this audiobook were written using a web-based **Integrated Development Environment** (or **IDE**) from [replit.com](#). A free replit.com account is sufficient to follow the examples. To use replit.com, all that is required is a modern web browser and a replit.com account. The code has also been tested on macOS using Visual Studio Code, although it should work with any code editor and properly configured operating system. Code examples are provided in both Node.js/JavaScript and Python. For Node.js, version 12.16.1 is used and for Python, version 3.8.2 is used.

All of the code examples will require an OpenAI API Key and access to the OpenAI API. You can request access to the OpenAI API by visiting [open AI.com/API](#).

Download the example code files

You can download the example code files for this audiobook from GitHub at [github.com/PacktPublishing/Exploring-GPT-3](#). In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Chapter 1

Technical requirements

This chapter requires you to have access to the **OpenAI Application Programming Interface (API)**. You can register for API access by visiting <https://openai.com/>.

Figures

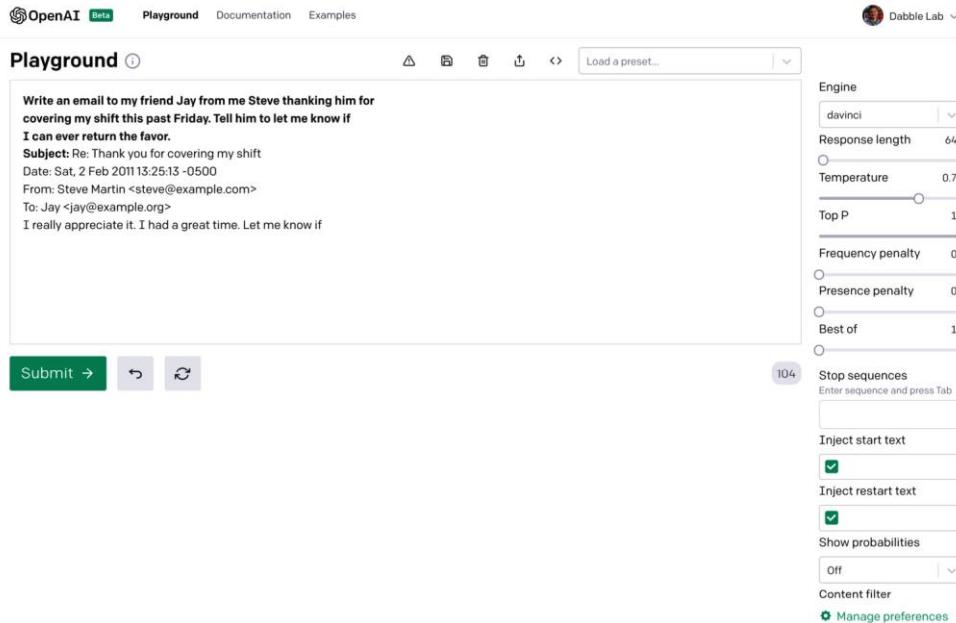


Figure 1.1 – Zero-shot prompt example

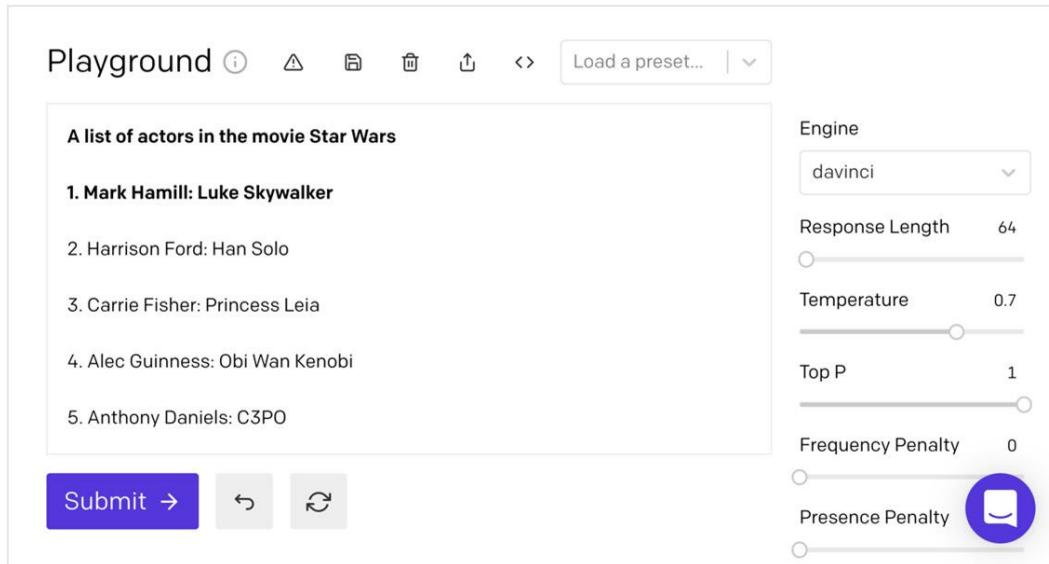


Figure 1.2 – One-shot prompt example

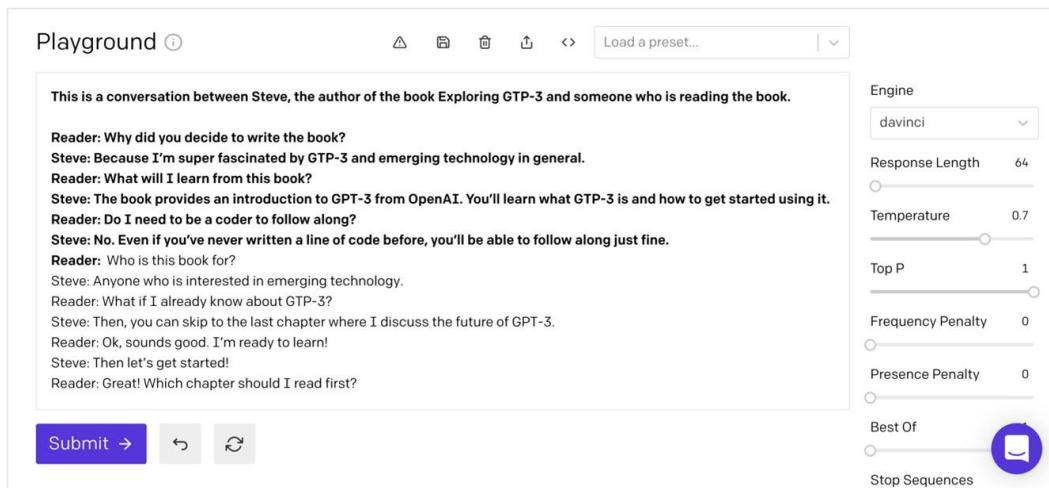


Figure 1.3 – Few-shot prompt example

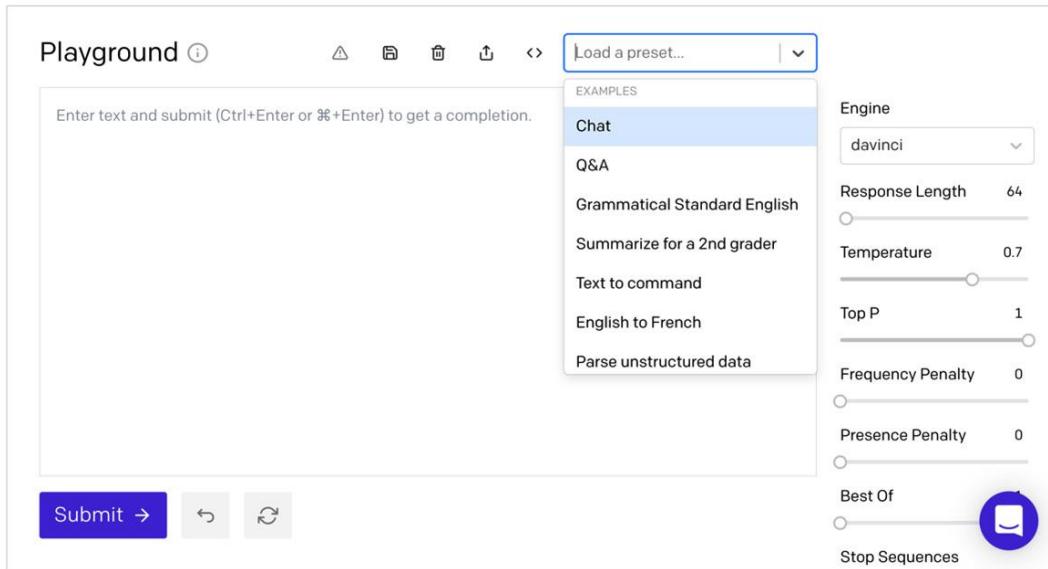


Figure 1.4 – Presets

The screenshot shows the OpenAI documentation for "Classification". The left sidebar contains a navigation menu with sections such as Introduction, Key Concepts, **Prompt Design 101** (which is currently selected and highlighted in green), Engines, Instruct Series, Content Filter, Examples, Summarization, Classification, Idea Generation, Developer Quickstart, API Keys, Making Requests, Python Bindings, Community Libraries, API Reference, Authentication, and List Engines. The main content area has a heading "Classification". It includes a text block: "To create a text classifier with the API we provide a description of the task and provide a few examples. In this demonstration we show the API how to classify the sentiment of Tweets." Below this is a code block showing several tweets and their corresponding sentiment classifications. At the bottom of the content area is a link "Open this example in Playground". A purple circular icon with a white speech mark is located in the bottom right corner of the content area.

```
This is a tweet sentiment classifier
Tweet: "I loved the new Batman movie!"
Sentiment: Positive
###
Tweet: "I hate it when my phone battery dies."
Sentiment: Negative
###
Tweet: "My day has been 👍"
Sentiment: Positive
###
Tweet: "This is the link to the article"
Sentiment: Neutral
###
Tweet: "This new music video blew my mind"
Sentiment:
```

Figure 1.5 – OpenAI documentation provides prompt examples

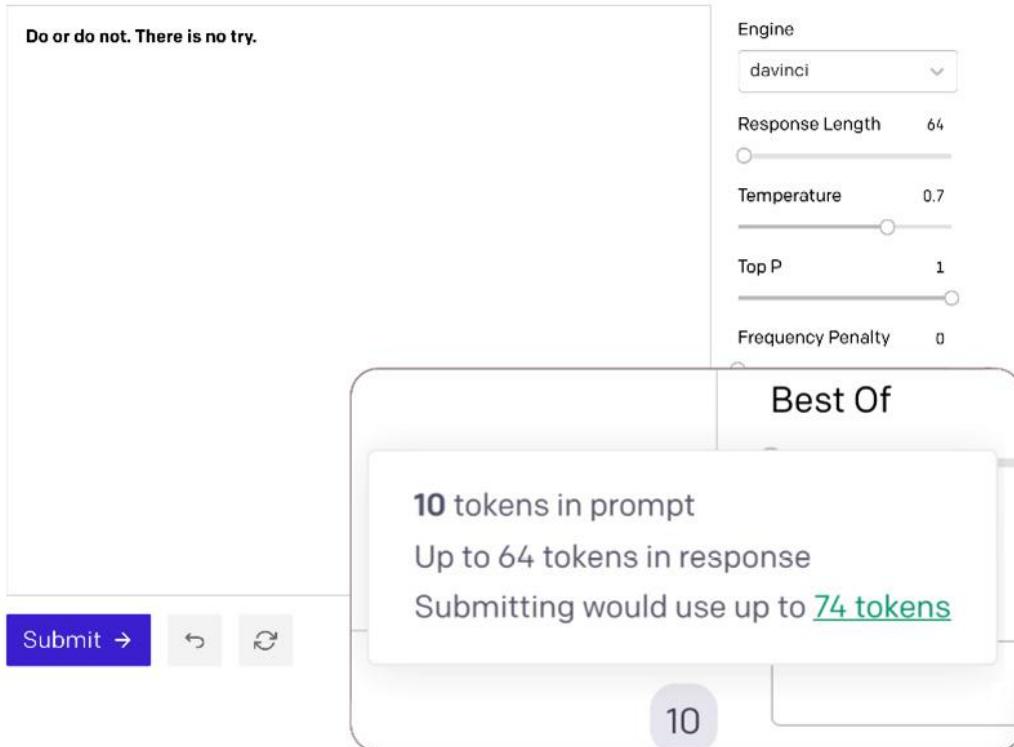


Figure 1.6 – Token count

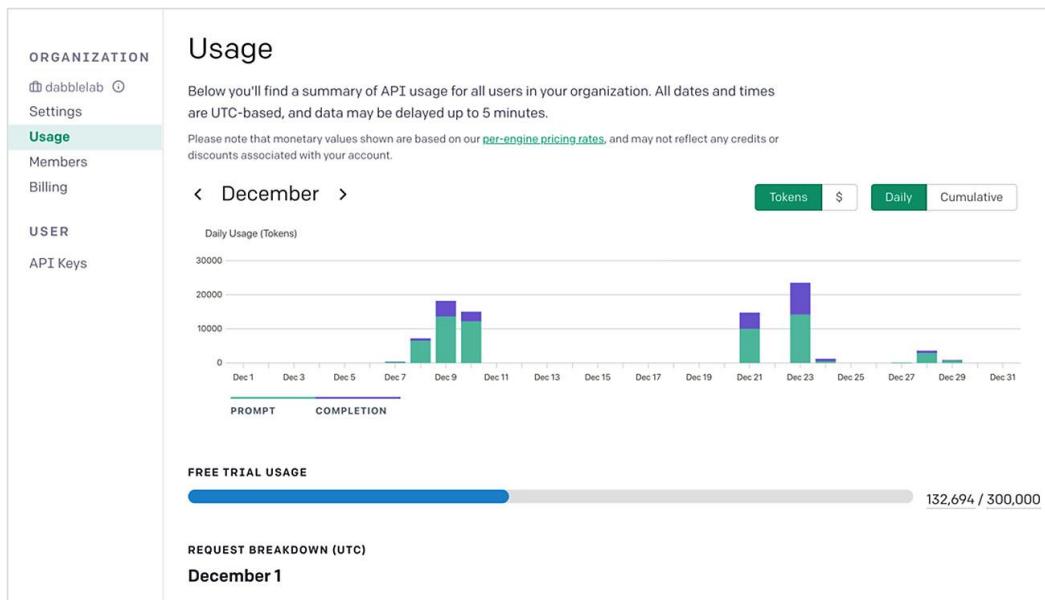


Figure 1.7 – Usage statistics

Per-model prices

The API offers multiple models with different capabilities and price points. Davinci is the most powerful model, while Ada is the fastest.

Prices are per 1,000 tokens. You can think of tokens as pieces of words, where 1,000 tokens is about 750 words. This paragraph is 35 tokens.

MODEL	PRICE PER 1K TOKENS
Davinci	\$0.0600
Curie	\$0.0060
Babbage	\$0.0012
Ada	\$0.0008

Figure 1.8 – Model pricing

Chapter 2

Technical requirements

This chapter requires you to have access to the OpenAI API. You can register for API access by visiting <https://openai.com>.

Figures

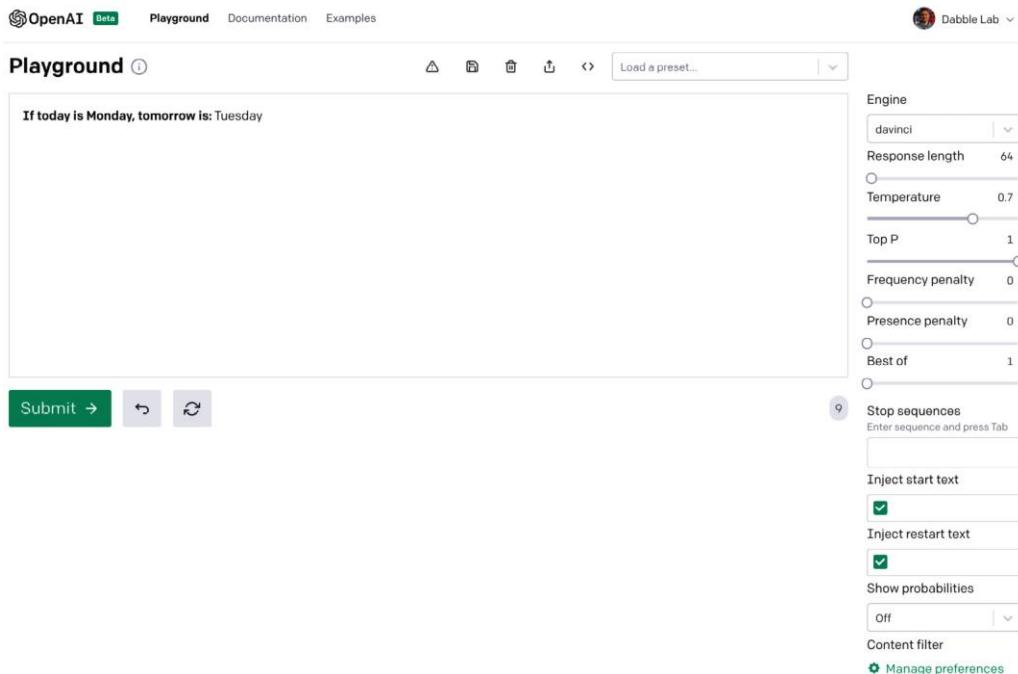


Figure 2.1 – Playground window

Playground ⓘ

Load a preset... | ↴

The following is a list of tips for first-time home buyers

1. Be sure to have a realistic budget . Make a list of how much you can afford to pay per month for your mortgage, including property taxes, insurance, maintenance, heating costs, and other expenses.
2. Don't be in a hurry . Don't buy a house just because it's available. It's a good idea to wait a year or two and save a larger down payment.
3. Shop for a mortgage lender . Mortgage lenders are not all the same. Shop around to find the one that offers the best interest rate and terms.

Submit → ⏪ ⏹ 145

Engine: davinci

Response Length: 64

Temperature: 0.7

Top P: 1

Frequency Penalty: 0

Presence Penalty: 0

Figure 2.2 – Text generation example – tips for first-time home buyers

Playground ⓘ

Load a preset... | ↴

Maker day 3D printer project ideas

1. GoPro Mount
A mount for a GoPro camera that mounts the camera on a mountain bike
2. Simple Door Knocker
A door knocker that can be programmed to say things or play sounds when someone knocks
3. Stand for iPad

Submit → ⏪ ⏹ 99

Engine: davinci

Response Length: 64

Temperature: 0.7

Top P: 1

Frequency Penalty: 0

Presence Penalty: 0

Figure 2.3 – Text generation example – 3D print project ideas

Playground ⓘ

Load a preset... | ↴

The following is a conversation with a customer support AI assistant. The assistant is helpful, creative, clever, and very friendly.

Customer: Hello, can you help me?

AI: I can sure try. I'm an AI support assistant and I'm here to help!

Customer: I have two problems. The first is that my computer is running slow and I need to clear some space on my hard drive. What should I do?

AI: That is a problem a lot of people have. I am recommending that you delete some old files. Any files that are old and unused can be deleted

Submit → ⏪ ⏹ 128

Figure 2.4 – Text generation example – customer support AI assistant

The following is a list of companies and the categories they fall into

- Cisco – Technology, Networking, Enterprise Software
- AT&T – Telecom, Technology, Conglomerate
- United Airlines – Aviation, Transportation
- Nvidia – Technology, Computing, Semiconductors
- McDonalds – Food, Fast Food, Restaurants
- Canon – Technology, Printing, Cameras
- HP – Computers, Printers, Hardware
- GE – Conglomerate, Aviation, Healthcare, Energy
- IBM – Technology, Computing, Hardware

Submit → ↺ ↻ 122 Engine: davinci Response Length: 64 Temperature: 0.7 Top P: 1 Frequency Penalty: 0 Presence Penalty: 0

Figure 2.5 – Text generation example – list generation

words that rhyme have similar sounding endings

- q: what rhymes with "cat"
a: bat, hat, mat
- q: what rhymes with "small"
a: tall, wall, call
- q: what rhymes with "pig"
a: big, dig, fig
- q: what rhymes with "dog"
a: log, bog, frog
- q: what rhymes with "duck"

Submit → ↺ ↻ 137 Engine: davinci Response Length: 64 Temperature: 0.3 Top P: 1 Frequency Penalty: 0 Presence Penalty: 0 Best Of: Stop Sequences

Figure 2.6 – Text generation example – quiz generation

The screenshot shows the OpenAI Playground interface. In the center, there is a text input area containing three paragraphs about quantum mechanics and classical physics. Below the text area are three buttons: 'Submit →', a back arrow, and a refresh arrow. To the right of the text area is a sidebar titled 'Engine' with dropdown menus for 'davinci' and '64'. It also includes sliders for 'Response Length' (64), 'Temperature' (0.3), 'Top P' (1), and checkboxes for 'Frequency Penalty' (0), 'Presence Penalty' (0), 'Best Of' (1), and 'Stop Sequences' (with a text input field). A small number '185' is displayed near the stop sequences button.

Figure 2.7 – Text generation example – tl;dr: summary

The screenshot shows the OpenAI Playground interface. In the center, there is a text input area containing two paragraphs about ownership and a one-sentence summary. Below the text area are three buttons: 'Submit →', a back arrow, and a refresh arrow. To the right of the text area is a sidebar with sliders for 'Frequency Penalty' (0), 'Presence Penalty' (0), 'Best Of' (1), and a 'Stop Sequences' section with a text input field. A small number '172' is displayed near the stop sequences button. There are also checkboxes for 'Inject Start Text' and 'Inject Restart Text', and a 'Show Probabilities' button set to 'Off'.

Figure 2.8 – Text generation example – one-sentence summary

The screenshot shows the OpenAI Playground interface. In the center, there is a text input area containing a paragraph about the Milky Way and a note about rephrasing it. Below the text area are three buttons: 'Submit →', a back arrow, and a refresh arrow. To the right of the text area is a sidebar with sliders for 'Response Length' (64), 'Temperature' (0.3), 'Top P' (1), and checkboxes for 'Frequency Penalty' (0), 'Presence Penalty' (0), 'Best Of' (1), and 'Stop Sequences' (with a text input field). A small number '331' is displayed near the stop sequences button.

Figure 2.9 – Text generation example – grade-level summarization

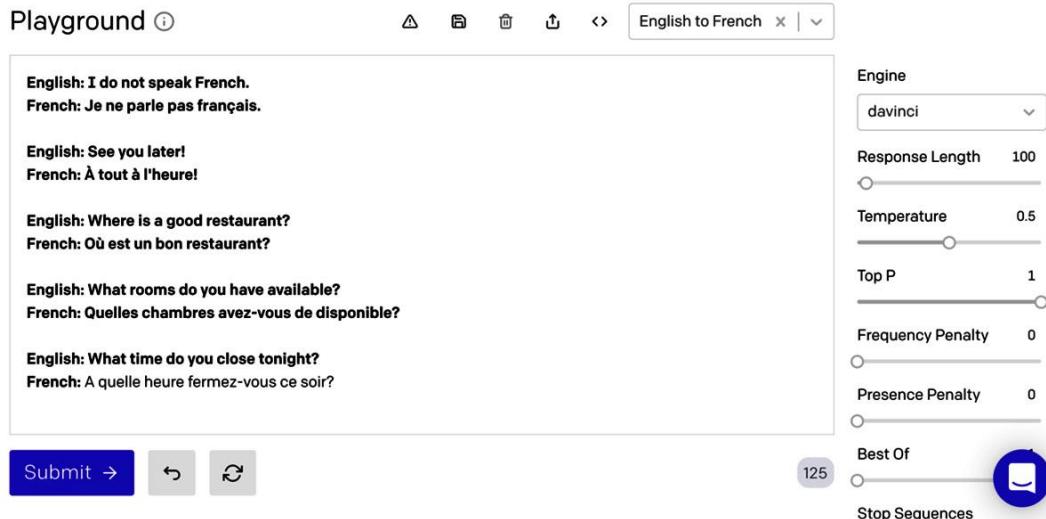


Figure 2.10 – Text generation example – translation from English to French

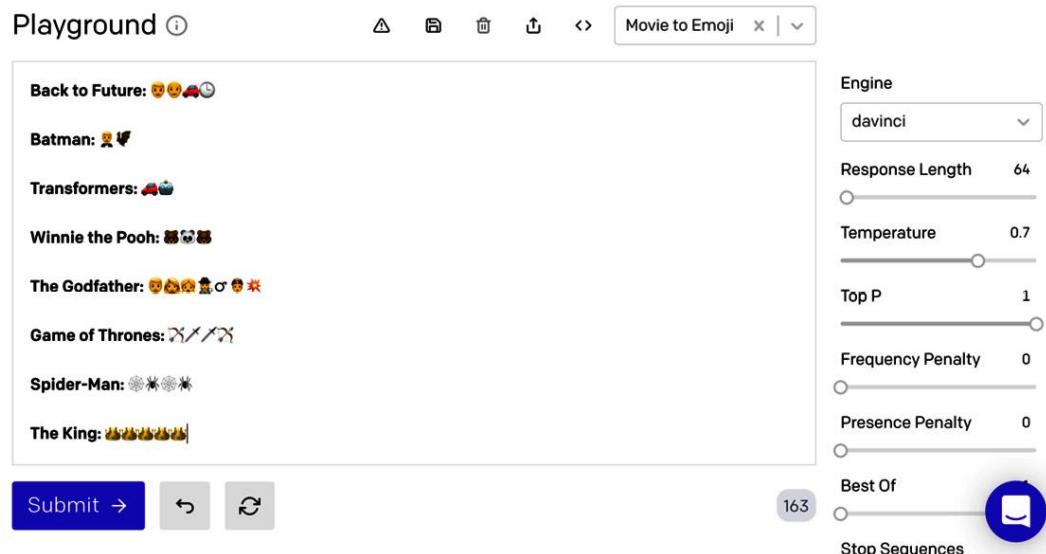


Figure 2.11 – Text generation example – conversion from text to emoji

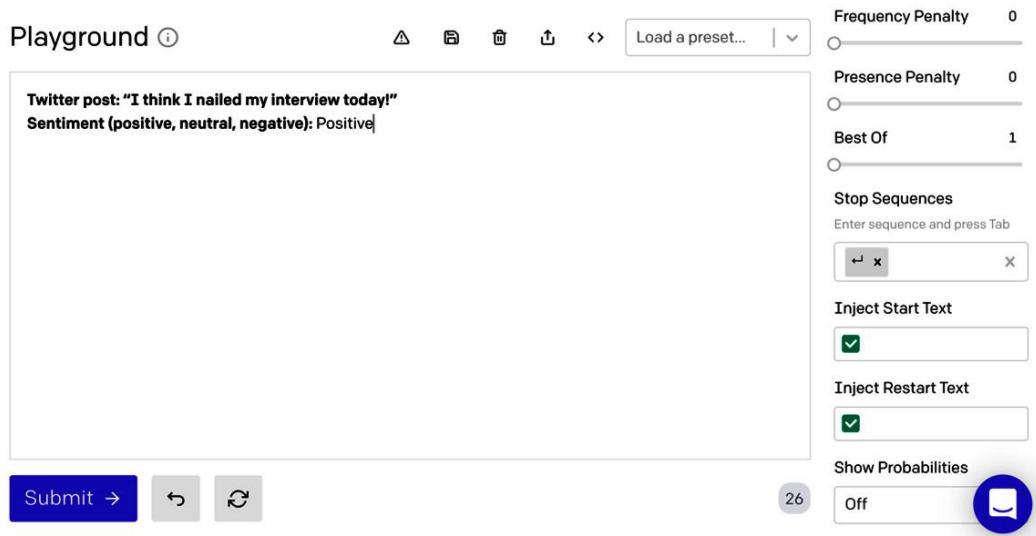


Figure 2.12 – Text generation example – zero-shot classification

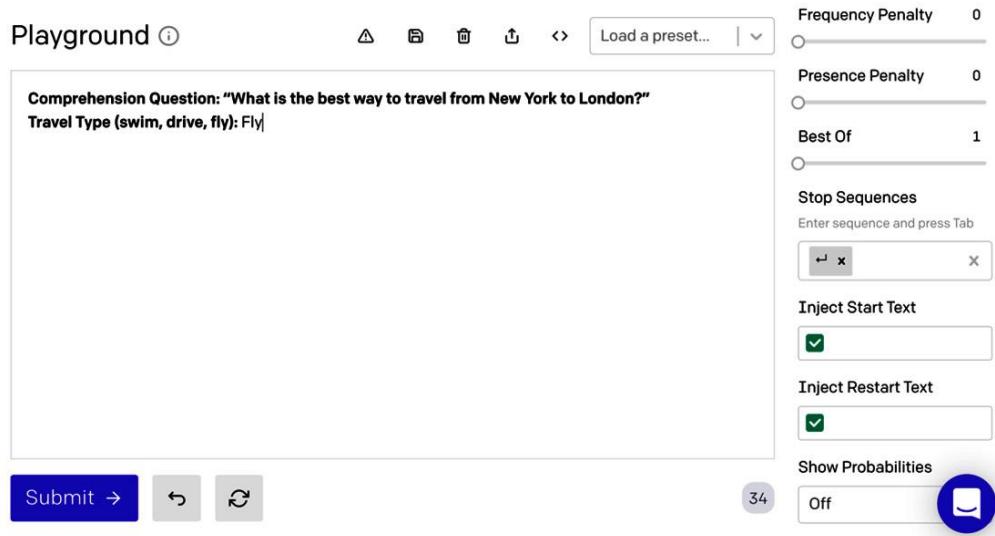


Figure 2.13 – Text generation example – zero-shot classification

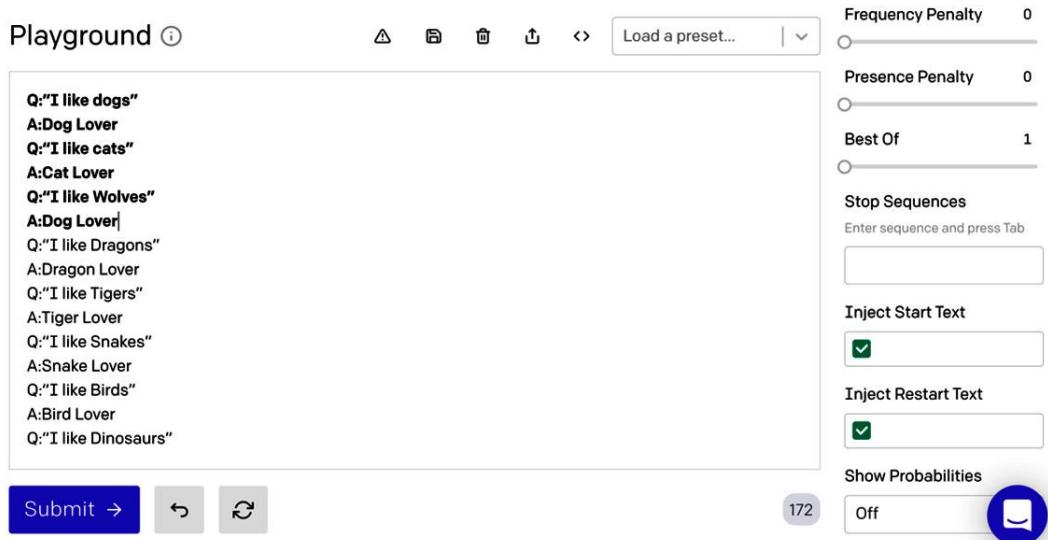


Figure 2.14 – Text generation example – few-shot classification

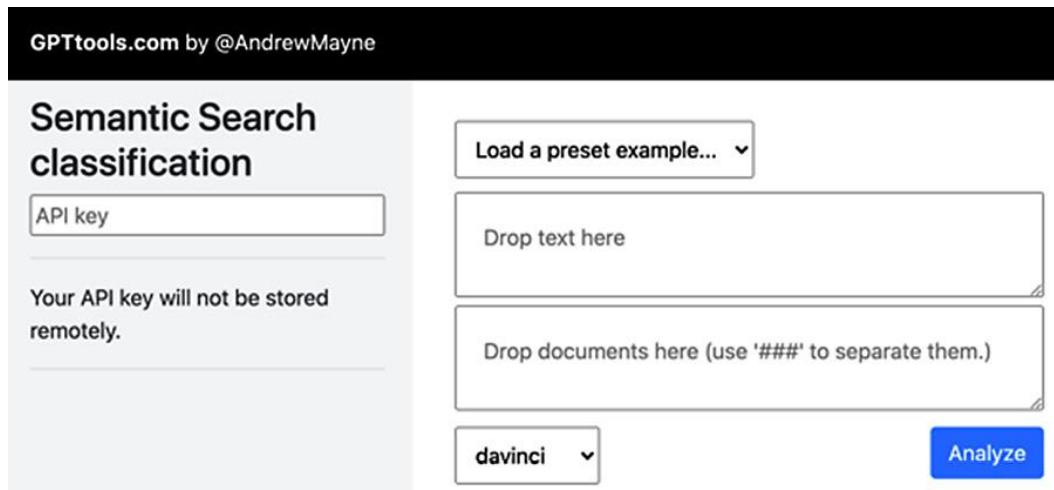


Figure 2.15 – The Semantic Search tool

The screenshot shows the 'API Keys' section of the OpenAI API Keys page. On the left, a sidebar lists 'ORGANIZATION' options: dabblelab (with a refresh icon), Settings, Usage, Members, and Billing. Below that is a 'USER' section with 'API Keys' selected. The main content area is titled 'Default Organization' and contains a note about selecting a default organization for API requests. A dropdown menu is set to 'dabblelab'. A note at the bottom explains that users can also specify an organization for each request and links to 'Authentication'.

This is your **Secret API Key**. Do not share this key with others, or expose it in the browser or other client-side code.

sk-vwccn [REDACTED]

⟳ Rotate Key

Default Organization

If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API key above.

dabblelab

Note: You can also specify which organization to use for each API request. See [Authentication](#) to learn more.

Figure 2.16 – The OpenAI API Keys page (with the API key intentionally blurred out)

The screenshot shows a 'Semantic Search classification' tool. It has a sidebar with the text 'Your API key will not be stored remotely.' At the top right is a dropdown menu set to 'Movie review sentiment'. Below it is a text input field containing a movie review: 'The script of this movie was probably found in a hair-ball coughed up by an old cat. Totally lame visual effects.' To the right, a blue box displays the sentiment analysis: 'The overall sentiment of the review is negative.' Below the input field are three lines of code: 'The overall sentiment of the review is negative.', '###', and 'The overall sentiment of the review is positive.' At the bottom are two buttons: a dropdown menu set to 'davinci' and a blue 'Analyze' button.

Semantic Search classification

Your API key will not be stored remotely.

Movie review sentiment

The script of this movie was probably found in a hair-ball coughed up by an old cat. Totally lame visual effects.

The overall sentiment of the review is negative.

###

The overall sentiment of the review is positive.

davinci

Analyze

Figure 2.17 – Text generation example – Semantic Search tool

Chapter 3

Technical requirements

This chapter requires that you have access to the **OpenAI API**. You can request access by visiting <https://openai.com/>.

Links

- <https://beta.openai.com/docs/introduction/prompt-design-101>

Figures

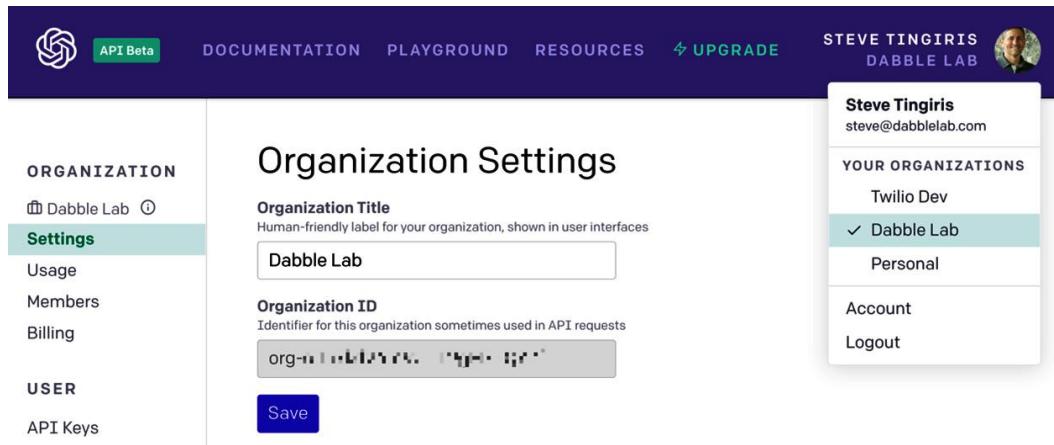


Figure 3.1 – Switching between organizations

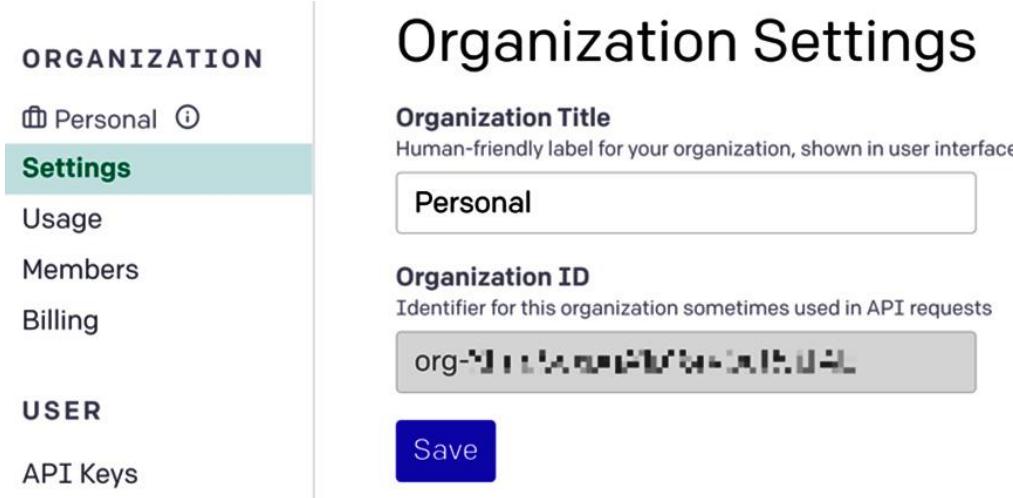


Figure 3.2 – Personal organization

The following screenshot shows the pricing per engine at the time this book was published. Again, the pricing could change at any time, so be sure to verify the current pricing as it may very well have changed:

Pricing	Go to Billing										
Usage Quotas											
FAQ											
<p>Our approach to pricing offers simplicity and flexibility: you pay only for the resources you use.</p> <p>Our API offers multiple engine types at different price points. Each engine has a spectrum of capabilities, with davinci being the most capable and ada the fastest. Requests to these different engines are priced differently.</p>											
<p>Trial: To explore and experiment with the API, all new users get 300,000 free tokens for the davinci engine, or the equivalent across other engines, which expire after 3 months.</p>											
<p>After the trial period, usage will be billed at the end of each calendar month for the resources used during that month at the following rates:</p>											
<table border="1"><thead><tr><th>Engine</th><th>Cost / 1K Tokens</th></tr></thead><tbody><tr><td>davinci</td><td>\$0.06</td></tr><tr><td>curie</td><td>\$0.006</td></tr><tr><td>babbage</td><td>\$0.0012</td></tr><tr><td>ada</td><td>\$0.0008</td></tr></tbody></table>		Engine	Cost / 1K Tokens	davinci	\$0.06	curie	\$0.006	babbage	\$0.0012	ada	\$0.0008
Engine	Cost / 1K Tokens										
davinci	\$0.06										
curie	\$0.006										
babbage	\$0.0012										
ada	\$0.0008										

Figure 3.3 – Pricing

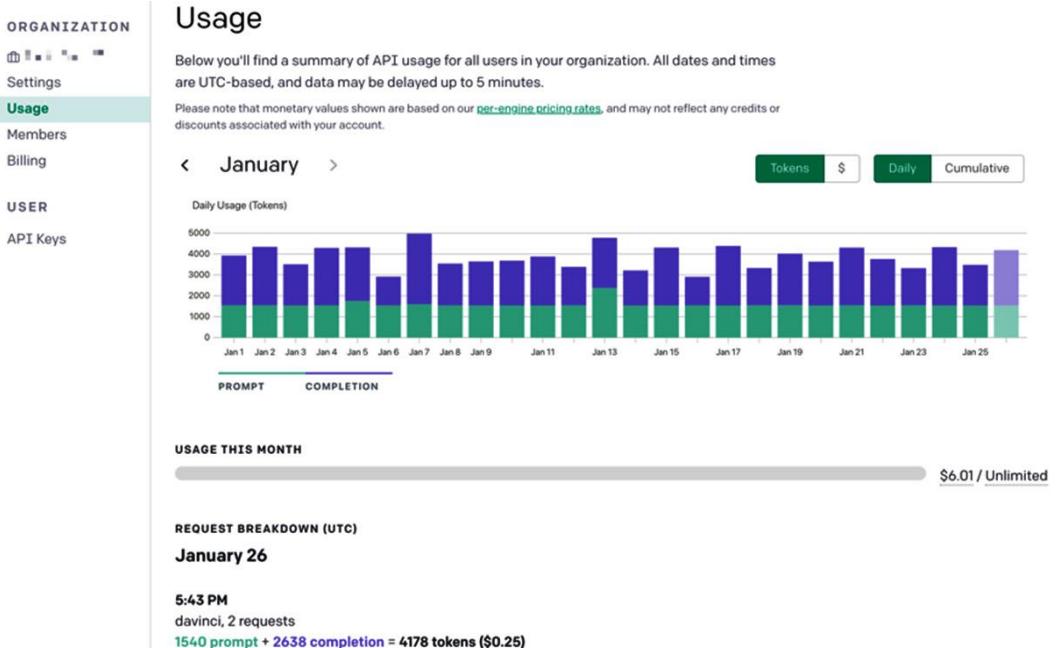


Figure 3.4 – Usage reporting

The screenshot shows the 'Members' section of an organization's dashboard. On the left, a sidebar lists 'ORGANIZATION' (Dabble Lab, Settings, Usage, **Members**, Billing) and 'USER' (API Keys). The main area is titled 'Members' and contains a table with five rows:

Profile	Role	Action
AL	Reader	Remove
SO	Reader	Remove
Owner	Owner	Leave
K	Invite pending	Delete

A blue 'Invite' button is located at the top right.

Figure 3.5 – Member management

The screenshot shows the 'Playground' settings interface. On the left, a large text input field says 'Enter text and submit (Ctrl+Enter or ⌘+Enter) to get a completion.' Below it are 'Submit →' and two small buttons. To the right is a vertical bar with the word 'SETTINGS' written across it. On the far right are various configuration options:

- Engine:** dropdown set to 'davinci'
- Response Length:** slider set to 64
- Temperature:** slider set to 0.7
- Top P:** slider set to 1
- Frequency Penalty:** slider set to 0
- Presence Penalty:** slider set to 0
- Best Of:** slider set to 1
- Stop Sequences:** text input field
- Inject Start Text:** checked checkbox
- Inject Restart Text:** checked checkbox
- Show Probabilities:** dropdown set to 'Off' with a speaker icon

Figure 3.6 – Playground settings

Playground ⓘ

The following is a list of items classified as a tool, food, clothing, or something else

Cake: Food
Pants: Clothing
Car: Other
Pliers: Tool
Shirt: Clothing
Hammer: Tool
Apple: Food
Airplane: Other

DAVINCI

Classification x | v

Engine: davinci

Response Length: 6

Temperature: 0

Top P: 1

Frequency Penalty: 0

Presence Penalty: 0

Best Of: 1

Stop Sequences: 

Submit → ⏪ ⏹ 59

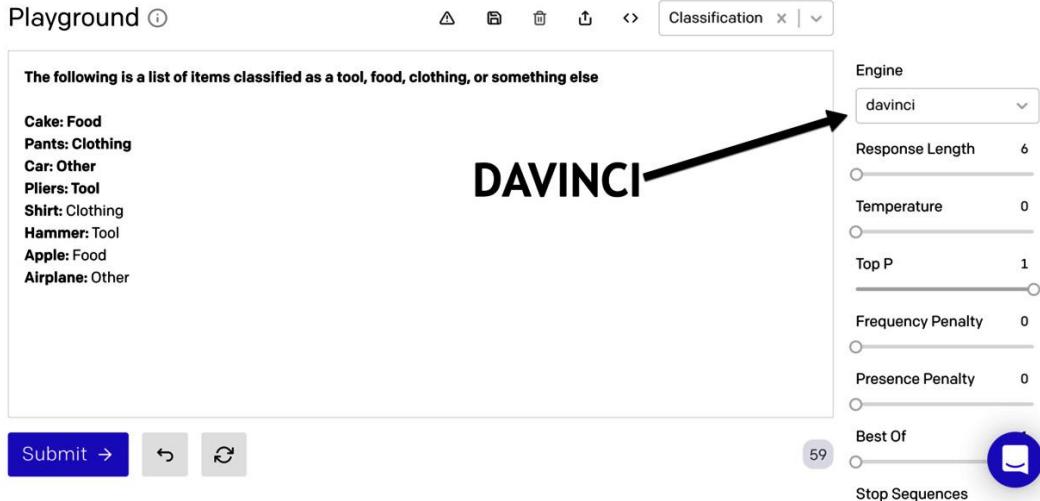


Figure 3.7 – Classification example with the davinci engine

Playground ⓘ

The following is a list of items classified as a tool, food, clothing, or something else

Cake: Food
Pants: Clothing
Car: Other
Pliers: Tool
Socks: Clothing
Pliers: Tool
Hamburger: Food
House: Other

ADA

Classification x | v

Engine: ada

Response Length: 6

Temperature: 0

Top P: 1

Frequency Penalty: 0

Presence Penalty: 0

Best Of: 1

Stop Sequences: 

Submit → ⏪ ⏹ 60

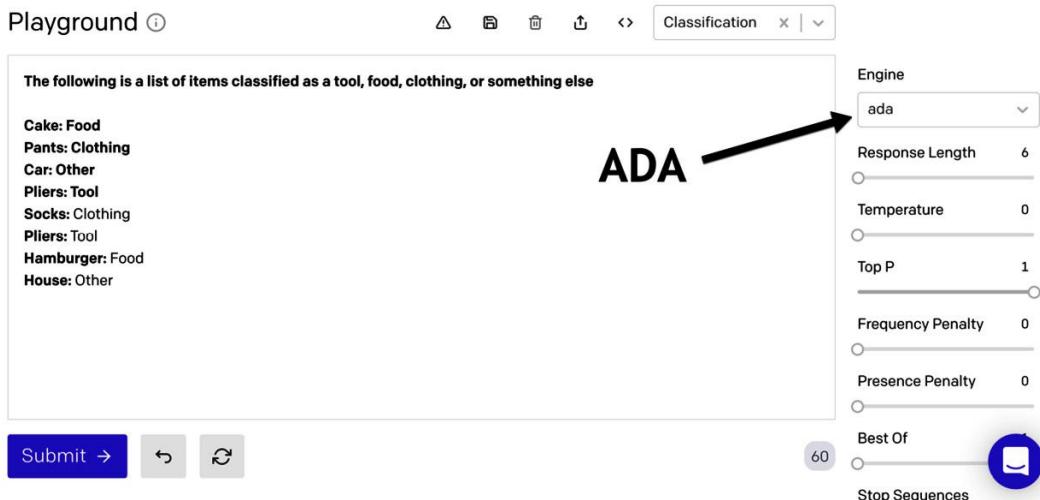


Figure 3.8 – Classification example with the ada engine

Playground ⓘ

Once upon a time|

Submit → ⏪ ⏴

Engine
davinci

Response Length 64

Temperature 0.7

Top P 1

Frequency Penalty 0

Presence Penalty 0 

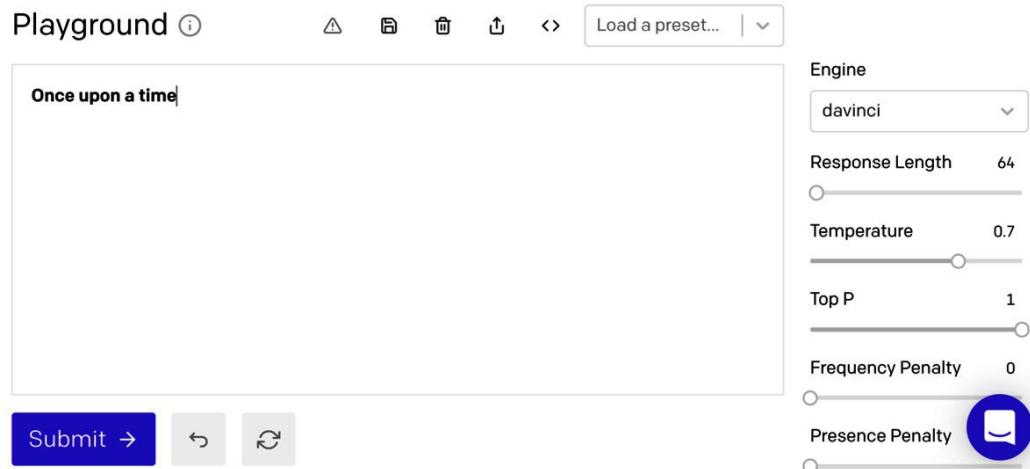


Figure 3.9 – Temperature example 1

Playground ⓘ

A robot may not injure

Submit → ⏪ ⏴

Engine
davinci

Response Length 64

Temperature 0.7

Top P 1

Frequency Penalty 0

Presence Penalty 0 

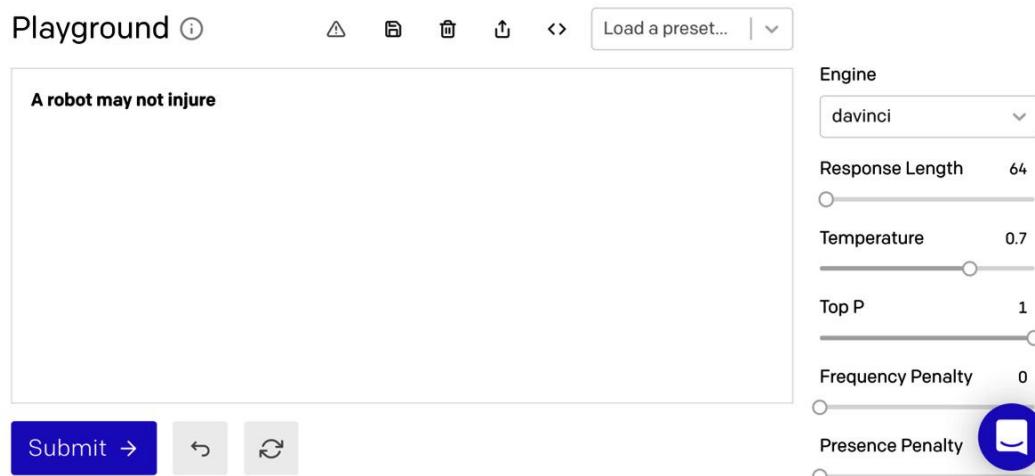


Figure 3.10 – Temperature example 2

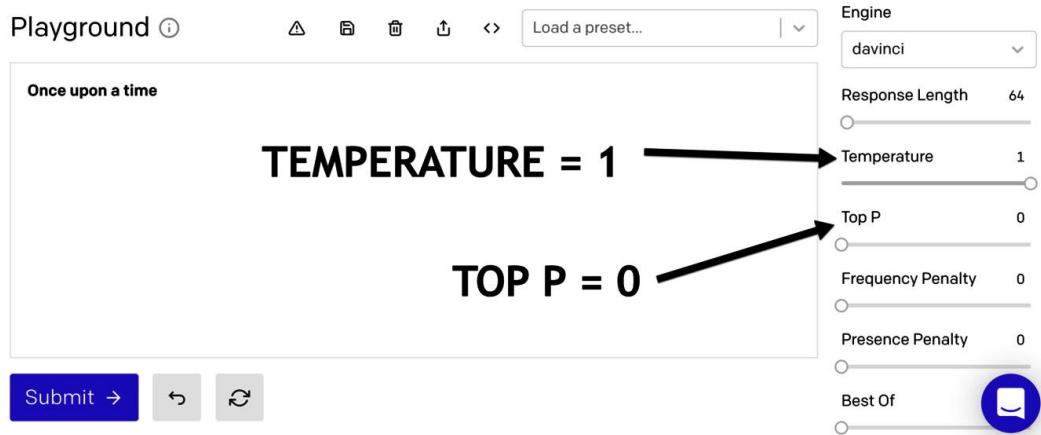


Figure 3.11 – Top P example

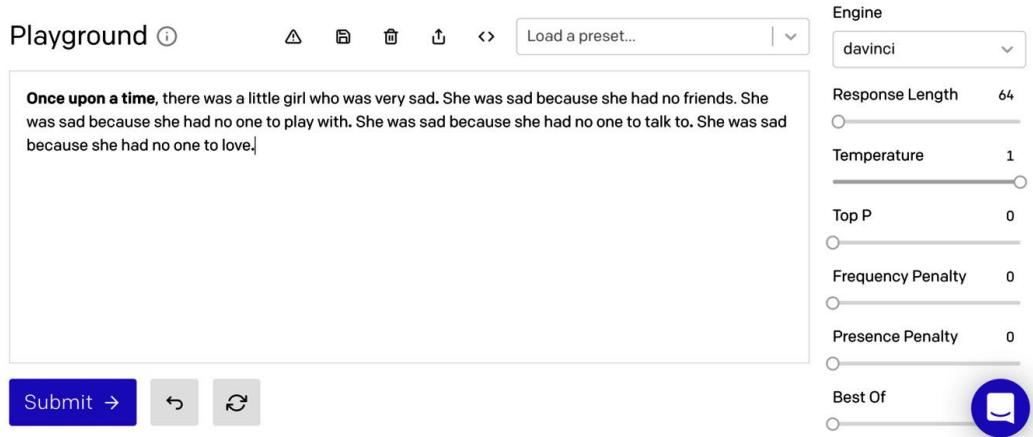


Figure 3.12 – Frequency and presence example 1

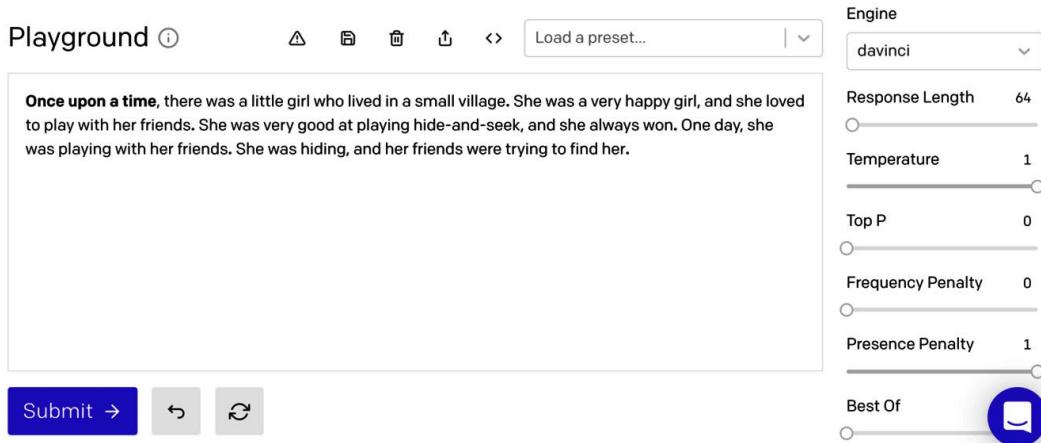


Figure 3.13 – Frequency and presence example 2

The screenshot shows the 'Playground' interface with the following configuration:

- Parse unstructured data** button is visible.
- Frequency Penalty**: 0
- Presence Penalty**: 0
- Best Of**: 1
- Stop Sequences**: An input field containing the sequence `x`.
- Inject Start Text**: A checked checkbox.

The main text area displays a story about fruits on planet Goocrux, followed by a table of fruit data:

	Fruit	Color	Flavor
1	Neoskizzles	Purple	Sweet
2	Loheckles	Grayish blue	Tart

A large black arrow points from the text area towards the 'Stop Sequences' section.

Figure 3.14 – Stop sequence

The screenshot shows the 'Playground' interface with the following configuration:

- Load a preset...** dropdown menu.
- Best Of**: 1
- Stop Sequences**: An input field.
- Inject Start Text**: A checked checkbox.
- Inject Restart Text**: A checked checkbox.
- Show Probabilities**: Off

The main text area shows a conversation:

Human: I'm feeling sad. Can you cheer me up?
Assistant: Sure. What would you like to do?
Human: I want to go to the beach.
Assistant: I can take you to the beach.

At the bottom are 'Submit' and 'Cancel' buttons, and a message count of 102.

Figure 3.15 – Default settings

The screenshot shows the 'Playground' interface with the following configuration:

- Load a preset...** dropdown menu.
- Best Of**: 1
- Stop Sequences**: An input field containing `Human: x`.
- Inject Start Text**: A checked checkbox with the value `Human: <--AI:`.
- Inject Restart Text**: A checked checkbox with the value `Human:`.
- Show Probabilities**: Off

The main text area shows a conversation:

Human: I'm feeling sad. Can you cheer me up?
AI: I'm sorry to hear that. I can't make you happy, but I can help you find happiness.
Human:

At the bottom are 'Submit' and 'Cancel' buttons, and a message count of 68.

Figure 3.16 – Using Stop Sequences, Inject Start Text, and Inject Restart Text together

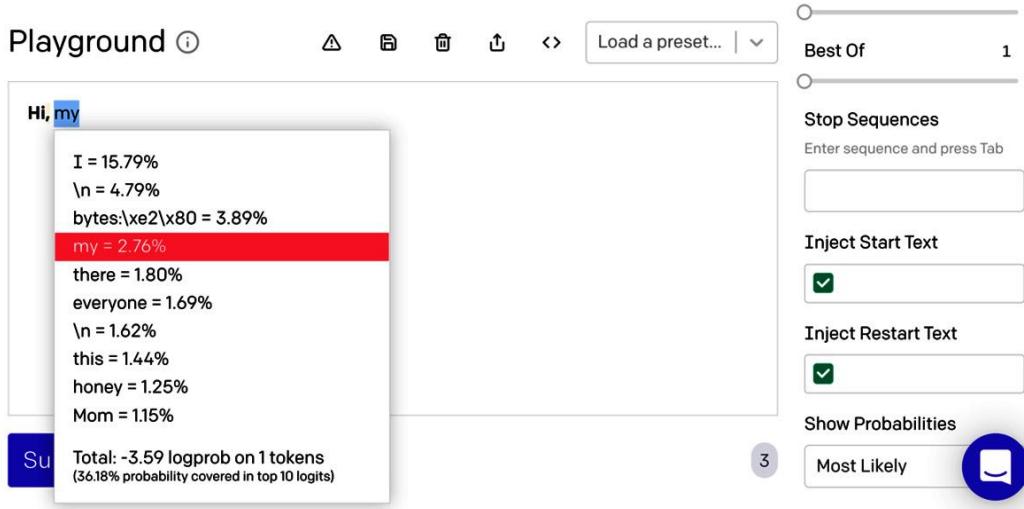


Figure 3.17 – Show Probabilities – most likely tokens

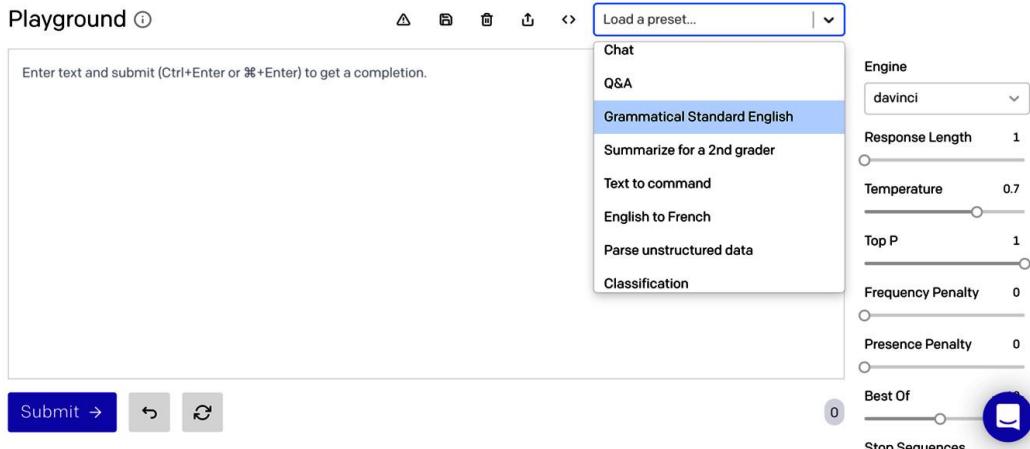


Figure 3.18 – Presets

Prompts

Prompt 3.1

Question: Who is Batman?

Answer: Batman is a fictional comic book character.

Question: What is torsalplexity?

Answer: ?

Question: What is Devz9?

Answer: ?

Question: Who is George Lucas?

Answer: George Lucas is American film director and producer famous for creating Star Wars.

Question: What is the capital of California?

Answer: Sacramento.

Question: What orbits the Earth?

Answer: The Moon.

Question: Who is Fred Rickerson?

Answer: ?

Question: What is an atom?

Answer: An atom is a tiny particle that makes up everything.

Question: Who is Alvan Muntz?

Answer: ?

Question: What is Kozar-09?

Answer: ?

Question: How many moons does Mars have?

Answer: Two, Phobos and Deimos.

Question:

Prompt 3.2

The following is a prompt classifying items as a tool, food, clothing, or something else:

The following is a list of items classified as a tool, food, clothing, or something else

Cake: Food

Pants: Clothing

Car: Other

Pliers: Tool

Prompt 3.3

The following is a conversation with an AI assistant. The assistant is helpful, creative, clever, and very friendly.

Human: I'm feeling sad. Can you cheer me up?

Prompt 3.4

Non-standard English: Please provide me with a short brief of the design you're looking for and that'd be nice if you could share some examples or project you did before.

Standard American English: Please provide me with a short brief of the design you're looking for and some examples or previous projects you've done would be helpful.

Non-standard English: If I'm stressed out about something, I tend to have problem to fall asleep.

Standard American English: If I'm stressed out about something, I tend to have a problem falling asleep.

Non-standard English: There is plenty of fun things to do in the summer when your able to go outside.

Standard American English: There are plenty of fun things to do in the summer when you are able to go outside.

Non-standard English: She no went to the market.

Standard American English: She didn't go to the market.

Prompt 3.5 – Text to command

Question: Ask Constance if we need some bread

Answer: send-msg `find constance` Do we need some bread?

Question: Send a message to Greg to figure out if things are ready for Wednesday.

Answer: send-msg `find greg` Is everything ready for Wednesday?

Question: Ask Ilya if we're still having our meeting this evening

Answer: send-msg `find ilya` Are we still having a meeting this evening?

Question: Contact the ski store and figure out if I can get my skis fixed before I leave on Thursday

Answer: send-msg `find ski store` Would it be possible to get my skis fixed before I leave on Thursday?

Question: Thank Nicolas for lunch

Answer: send-msg `find nicolas` Thank you for lunch!

Question: Tell Constance that I won't be home before 19:30 tonight – unmovable meeting.

Answer: send-msg `find constance` I won't be home before 19:30 tonight. I have a meeting I can't move.

Question:

Prompt 3.6 – Parse unstructured data

There are many fruits that were found on the recently discovered planet Goocrux. There are neoskizzles that grow there, which are purple and taste like candy. There are also loheckles, which are a grayish blue fruit and are very tart, a little bit like a lemon. Pounits are a bright green color and are more savory than sweet. There are also plenty of loopnovas which are a neon pink flavor and taste like cotton candy. Finally, there are fruits called glowls, which have a very sour and bitter taste which is acidic and caustic, and a pale orange tinge to them.

Please make a table summarizing the fruits from Goocrux

Fruit	Color	Flavor
neoskizzles	purple	candy
loheckles	grayish blue	tart
pounits	bright green	savory
loopnovas	neon pink	cotton candy
glowl	various	acidic and caustic

| Neoskizzles | Purple | Sweet |

| Loheckles | Grayish blue | Tart |

Chapter 4

Technical requirements

This chapter requires that you have access to the **OpenAI API**. You can request access by visiting <https://openai.com/>.

Links

- For a full list of the different status codes, you can visit <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>.
- You can learn more about create completion endpoint by visiting <https://beta.openai.com/docs/api-reference/create-completion-via-gel>.
- If you don't have CURL installed, you can download it from the official site at <https://curl.se/download.html>.

Figures

```
{"message": "success", "number": 7, "people": [{"craft": "ISS", "name": "Sergey Ryzhikov"}, {"craft": "ISS", "name": "Kate Rubins"}, {"craft": "ISS", "name": "Sergey Kud-Sverchkov"}, {"craft": "ISS", "name": "Mike Hopkins"}, {"craft": "ISS", "name": "Victor Glover"}, {"craft": "ISS", "name": "Shannon Walker"}, {"craft": "ISS", "name": "Soichi Noguchi"}]}
```

Figure 4.1 – Open-Notify API – JSON response



DOCUMENTATION PLAYGROUND RESOURCES ⚡ UPGRADE

STEVE TINGIRIS DABBLE LAB 

GET STARTED

- [Introduction](#)
- [Developer quickstart](#)
- [API keys](#)
- Making requests**
- [Python bindings](#)
- [Community libraries](#)
- [Engines](#)
- [Examples](#)
- [Going live](#)
- [Use case guidelines](#)
- [Safety best practices](#)

API REFERENCE

- [Introduction](#)
- [Authentication](#)
- [List engines](#)

Making requests

Copy the curl command below and paste it into your terminal to run your first request.

```
curl https://api.openai.com/v1/engines/davinci/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer sk-...e...n...d...t...o...n...l...k..." \
-d '{"prompt": "This is a test", "max_tokens": 5}'
```

This request queries the davinci engine to complete the text starting with a prompt of "*This is a test*". The `max_tokens` parameter sets an upper bound on how many tokens (which are the chunks of text that the API generates one at a time) so you'll get a response back like the following:

```
[{"id": "cmpl-GERzeJQ41vqPk8SkZu4XMIuR",
"object": "text_completion",
"created": 1586839808,
"model": "davinci:2020-05-03",
"choices": [{"
```



Figure 4.2 – CURL command in the OpenAI API docs

```
Last login: Fri Jan  1 10:57:12 on console
$ curl
curl: try 'curl --help' or 'curl --manual' for more information
$ curl --help
Usage: curl [options...] <url>
  --abstract-unix-socket <path> Connect via abstract Unix domain socket
  --alt-svc <file name> Enable alt-svc with this cache file
  --anyauth      Pick any authentication method
  --cert <file> CA certificate to verify peer against
  --capath <dir> CA directory to verify peer against
-E, --cert <certificate[:password]> Client certificate file and password
  --cert-status Verify the status of the server certificate
  --cert-type <type> Certificate file type (DER/PEM/ENG)
  --ciphers <list of ciphers> SSL ciphers to use
  --compressed Request compressed response
  --compressed-ssh Enable SSH compression
-K, --config <file> Read config from a file
  --connect-timeout <seconds> Maximum time allowed for connection
  --connect-to <HOST1:PORT1:HOST2:PORT2> Connect to host
-C, --continue-at <offset> Resumed transfer offset
-b, --cookie <data>filename> Send cookies from string/file
-c, --cookie-jar <filename> Write cookies to <filename> after operation
  --create-dirs Create necessary local directory hierarchy
  --crlf      Convert LF to CRLF in upload
  --crlfile <file> Get a CRL list in PEM format from the given file
-d, --data <data> HTTP POST data
  --data-ascii <data> HTTP POST ASCII data
  --data-binary <data> HTTP POST binary data
  --data-raw <data> HTTP POST data, '@' allowed
  --data-urlencode <data> HTTP POST data url encoded
  --delegation <LEVEL> GSS-API delegation permission
  --digest      Use HTTP Digest Authentication
-a, --disable   Disable .curlrc
  --disable-eprt Inhibit using EPRT or LPRT
  --disable-epsv Inhibit using EPSV
  --disallow-username-in-url Disallow username in url
  --dns-interface <interface> Interface to use for DNS requests
  --dns-ipv4-addr <address> IPv4 address to use for DNS requests
  --dns-ipv6-addr <address> IPv6 address to use for DNS requests
  --dns-servers <addresses> DNS server addrs to use
  --doh-url <URL> Resolve host names over DOH
-D, --dump-header <filename> Write the received headers to <filename>
  --egd-file <file> EGD socket path for random data
  --engine <name> Crypto engine to use
  --expect100-timeout <seconds> How long to wait for 100-continue
-f, --fail      Fail silently (no output at all) on HTTP errors
  --fail-early   Fail on first transfer error, do not continue
  --false-start  Enable TLS False Start
-F, --form <name=>content> Specify multipart MIME data
```

Figure 4.3 – Curl Help command

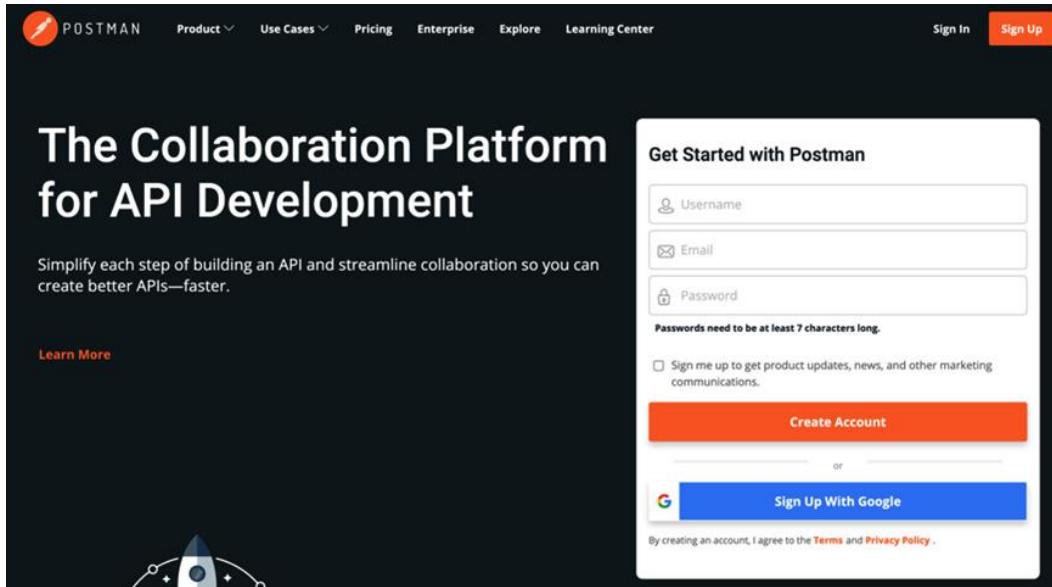


Figure 4.4 – Postman home page

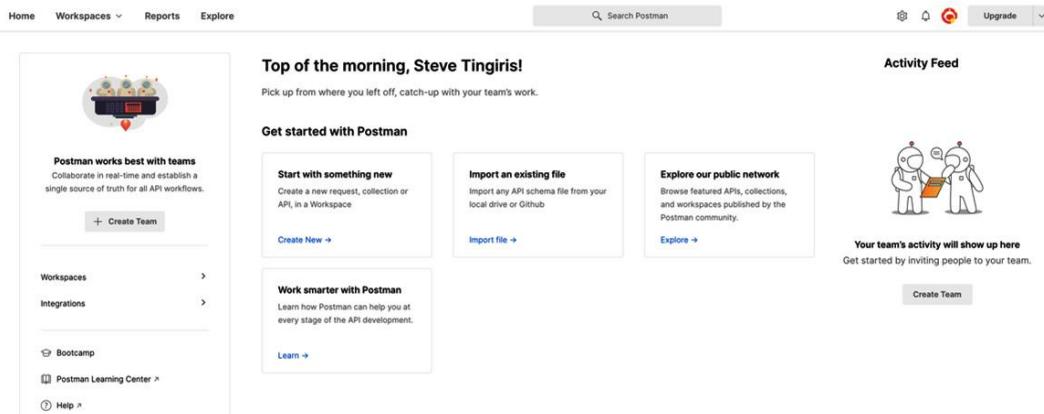


Figure 4.5 – Postman welcome screen

The screenshot shows the 'My Workspace' section of the Postman application. On the left, there's a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area has a heading 'My Workspace' with a sub-section 'In this workspace'. It displays four metrics: 0 Collections, 0 APIs, 0 Environments, and 0 Mock Servers. Below these metrics is a 'Activity' section showing a single entry: 'Steve Tingiris created this personal workspace 14 mins ago'. To the right, there are sections for 'Get started' (with links to Create a request, Create a collection, Create an API, and Create an environment) and 'Sharing' (set to 'Personal'). A note at the bottom says 'The end! You've seen all the activity for this workspace'.

Figure 4.6 – My Workspace

The screenshot shows a POSTMAN request to the URL `http://api.open-notify.org/astros.json`. The request method is GET. The response status is 200 OK, with a time of 522 ms and a size of 716 B. The response body is a JSON object containing the following data:

```
1 "message": "success",
2 "number": 11,
3 "people": [
4     {
5         "name": "Mike Hopkins",
6         "craft": "ISS"
7     },
8     {
9         "name": "Victor Glover",
10        "craft": "ISS"
11    },
12    {
13        "name": "Shannon Walker",
14        "craft": "ISS"
15    },
16    {
17        "name": "Soichi Noguchi",
18        "craft": "ISS"
19    }
20 ]
```

Figure 4.7 – Postman request results

The screenshot shows the Dabble Lab API Keys page. At the top right, there is a user profile for Steve Tingiris (steve@dabblelab.com) with options for 'YOUR ORGANIZATIONS' (Dabble Lab), 'Personal', 'Account', and 'Logout'. A red arrow points from the text 'API KEY' to the 'Secret API Key' field, which contains a long string of characters: `sk-vwccnGPPoMYBQpi2Hz3KAxQ8TINb2PofMPz48S3N`. Below this field is a 'Rotate Key' button.

Figure 4.8 – API Keys

The screenshot shows the OpenAI API documentation. On the left sidebar, under 'API REFERENCE', the 'Authentication' section is highlighted. In the main content area, the 'Authentication' section is titled 'Authentication'. It explains that the OpenAI API uses API keys for authentication and provides instructions for including the key in an Authorization header. A red arrow points from the text 'API KEY' to the 'Authorization' header example: `Authorization: Bearer sk-vwccnGPPoMYBQpi2Hz3KAxQ8TINb2PofMPz48S3N`. Below this, there is an 'Important' note about not sharing the key with users or exposing it to the browser. The 'Requesting organization' section at the bottom also mentions specifying an organization for API requests.

Figure 4.9 – API key in documentation

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, Reports, and Explore. Under 'My Workspace', it says 'You don't have any collections' and has a 'Create Collection' button. The main area shows a request for 'https://api.openai.com/v1/engines' using a GET method. The 'Params' tab is selected, showing a single parameter 'key' with a value of 'null'. Below the parameters, the response body is displayed in JSON format:

```
1
2
3
4
5
6
7
8
{
  "error": [
    {
      "code": null,
      "message": "You didn't provide an API key. You need to provide your API key in an Authorization header using Beamer auth (i.e. Authorization: Beamer YOUR_KEY), or as the password field (with blank username) if you're accessing the API from your browser and are prompted for a username and password. You can obtain an API key from https://beta.openai.com. Feel free to email support@openai.com if you have any questions."
    }
  ],
  "param": null,
  "type": "invalid_request_error"
}
```

The status bar at the bottom indicates 'Status: 401 Unauthorized'.

Figure 4.10 – API request without the API key

This screenshot shows the Postman interface with an environment named 'openai-dev' selected. The top navigation bar includes 'Invite', 'Upgrade', and environment selection buttons. Below the environment name, there are 'Save' and 'Edit' buttons. At the bottom, there are tabs for 'Tests', 'Settings', and 'Cookies', with 'Cookies' currently active. A large blue 'Send' button is prominently displayed.

Figure 4.11 – Postman with the environment set

The screenshot shows the Postman interface for a GET request to <https://api.openai.com/v1/engines>. The 'Authorization' tab is selected, showing a 'Bearer Token' input field with a placeholder: 'The authorization header will be automatically generated when you send the request.' A red circle highlights the status bar at the bottom right, which displays 'Status: 200 OK Time: 3.90'. The response body is a JSON object:

```

1
2   "object": "list",
3   "data": [
4     {
5       "id": "ada",
6       "object": "engine",
7       "created": null,
8       "max_replicas": null,
9       "owner": "openai",
10      "permissions": null,
11      "ready": true,
12      "ready_replicas": null,

```

Figure 4.12 – API request using the API key as a bearer token

The screenshot shows the Postman interface for a GET request to <https://api.openai.com/v1/engines>. The 'Headers' tab is selected, showing the following configuration:

KEY	VALUE	DESCRIPTION	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	Bearer sk-[REDACTED]			
<input checked="" type="checkbox"/> Host	<calculated when request is sent>			
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.28.0			
<input checked="" type="checkbox"/> Accept	*			
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br			
<input checked="" type="checkbox"/> Connection	keep-alive			

The 'Body' tab is also visible at the bottom, showing the same JSON response as in Figure 4.12.

Figure 4.13 – Authorization header with the API key as a bearer token

The screenshot shows the 'Organization' settings page for an organization named 'Dabble Lab'. The 'Organization Title' is set to 'Dabble Lab'. The 'Organization ID' field contains the value 'org-c1...'. A 'Save' button is visible at the bottom.

Figure 4.14 – Finding your organization ID

The screenshot shows a POST request to 'https://api.openai.com/v1/engines'. The 'Headers' tab is selected, showing the following configuration:

Key	Value	Description
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.28.0	
Accept	*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
OpenAI-Organization	{{OPENAI_ORGANIZATION_ID}}	

The response status is 200 OK, and the JSON response body is:

```
1 "object": "list",
2 "data": [
```

Figure 4.15 – Using the OpenAI-Organization HTTP header

```

$ curl https://api.openai.com/v1/engines \
>   -H 'Authorization: Bearer sk-' \
>   -H 'OpenAI-Organization: org-' \
{
  "object": "list",
  "data": [
    {
      "id": "ada",
      "object": "engine",
      "created": null,
      "max_replicas": null,
      "owner": "openai",
      "permissions": null,
      "ready": true,
      "ready_replicas": null,
      "replicas": null
    },
    {
      "id": "babbage",
      "object": "engine",
      "created": null,
      "max_replicas": null,
      "owner": "openai",
      "permissions": null,
      "ready": true,
      "ready_replicas": null,
      "replicas": null
    },
    {
      "id": "content-filter-alpha-c4",
      "object": "engine",
      "created": null,
      "max_replicas": null,
      "owner": "openai",
      "permissions": null,
      "ready": true,
      "ready_replicas": null,
      "replicas": null
    }
  ]
}

```

Figure 4.16 – Using CURL to call the List Engines endpoint

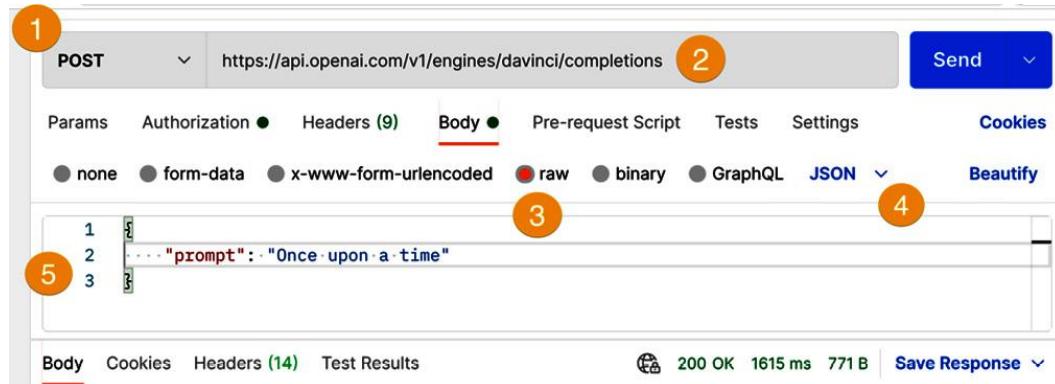


Figure 4.17 – Postman settings for the Completions endpoint

The screenshot shows a Postman interface with a POST request to `https://api.openai.com/v1/engines/davinci/completions`. The Body tab is selected, containing the JSON input:

```
1
2   ...
3     "prompt": "Once upon a time"
```

The response tab shows a status of 200 OK with a response body:

```
1 {
2   "id": "cmpl-2un9J1aeZKwImbKhWAdwwECZipXVA",
3   "object": "text_completion",
4   "created": 1619887973,
5   "model": "davinci:2020-05-03",
6   "choices": [
7     {
8       "text": " on the dangerous side of the tracks, the Wonderbolts were your guardians of",
9       "index": 0,
10      "logprobs": null,
11      "finish_reason": "length"
12    }
13  ]
14 }
```

Figure 4.18 – Postman response from the Completions endpoint

The screenshot shows a Postman interface with a POST request to `https://api.openai.com/v1/engines/davinci/completions`. The Body tab is selected, containing the JSON input:

```
1
2   ...
3     "prompt": "Once upon a time"
```

The response tab shows a status of 200 OK with a response body:

```
{"id": "cmpl-2un9J1aeZKwImbKhWAdwwECZipXVA", "object": "text_completion", "created": 1619887973, "model": "davinci:2020-05-03", "choices": [{"text": " on the dangerous side of the tracks, the Wonderbolts were your guardians of", "index": 0, "logprobs": null, "finish_reason": "length"}]}
```

Figure 4.19 – Postman response from the Completions endpoint – Raw

A screenshot of the Postman application interface. At the top, it shows a POST request to the URL <https://api.openai.com/v1/engines/davinci/completions>. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "prompt": "Once upon a time",
3   "max_tokens": 4
4 }
```

The status bar at the bottom indicates a 200 OK response with a time of 1615 ms and a size of 771 B.

Figure 4.20 – Postman response from the Completions endpoint with max_tokens

A screenshot of the Postman application interface. At the top, it shows a POST request to the URL <https://api.openai.com/v1/engines/davinci/completions>. The 'Body' tab is selected, showing a JSON payload with multiple prompts:

```
1 {
2   "prompt": ["The capital of California is:", "Once upon a time"],
3   "max_tokens": 7,
4   "temperature": 0,
5   "stop": "\n"
6 }
```

The status bar at the bottom indicates a 200 OK response with a time of 1733 ms and a size of 804 B.

Below the body, there is a preview section showing the resulting completion object:

```
1 {
2   "id": "cmpl-2unXXYTvidHwzUwLEUo59AGMHXkGN",
3   "object": "text_completion",
4   "created": 1619889475,
5   "model": "davinci:2020-05-03",
6   "choices": [
7     {
8       "text": " Sacramento",
9       "index": 0,
10      "logprobs": null,
11      "finish_reason": "stop"
12    },
13    {
14      "text": ", there was a little girl who",
15      "index": 1,
16      "logprobs": null,
17    }
18  ]
19 }
```

Figure 4.21 – Sending multiple prompts

The screenshot shows a Postman interface with a POST request to `https://api.openai.com/v1/engines/davinci/search`. The request body is a JSON object:

```

1 {
2   "documents": [
3     "plane",
4     "boat",
5     "spaceship",
6     "car"
7   ],
8   "query": "A vehicle with wheels"
9 }

```

The response status is 200 OK, and the response body is:

```

1 {
2   "object": "list",
3   "data": [
4     {
5       "object": "search_result",
6       "document": 0,
7       "score": 56.118
8     },
9     {
10       "object": "search_result",
11       "document": 1,
12       "score": 46.883
13     },
14     {
15       "object": "search_result",

```

Figure 4.22 – Semantic Search results

Code

Code 4.1 – The following is the HTTP method (GET) and URI for the List Engines endpoint:

```
get https://api.openai.com/v1/engines
```

Code 4.2 – The Retrieve Engine endpoint uses the HTTP get method and the following URI with one parameter, the engine ID:

```
get https://api.openai.com/v1/engines/{engine_id}
```

Code 4.3 – The Completions endpoint also uses the post method and requires an engine ID as a URI parameter:

```
post https://api.openai.com/v1/engines/{engine_id}/completions
```

Code 4.4 – The Semantic Search endpoint uses the post method and requires an engine ID as a URI parameter:

```
post https://api.openai.com/v1/engines/{engine_id}/search
```

Code 4.5 – The following is an example of a CURL command for calling the List Engines endpoint:

```
curl https://api.openai.com/v1/engines \
-H 'Authorization: Bearer {your-api-key}' \
-H 'OpenAI-Organization: {your-orgainzation-id}'
```

Code 4.6 – The following code block is an example of a JSON object from an OpenAI API response.

```
{
  "id": "cmpl-2T0IrOkctsOm8uVFvDDEmc1712U9R",
  "object": "text_completion",
  "created": 1613265353,
  "model": "davinci:2020-05-03",
  "choices": [
    {
      "text": ", there was a dog",
      "index": 0,
      "logprobs": null,
      "finish_reason": "length"
    }
  ]
}
```

Code 4.7 – Technically, the JSON body could just be an empty object (just a left and right curly brace, like `{ }`), but minimally, you'll want to include at least the `prompt` element with the value set to your `prompt` string, something like the following JSON example:

```
{"prompt": "Once upon a time"}
```

Code 4.8 – We'll use the following JSON to submit multiple prompts simultaneously and get back a completion for each:

```
{  
  "prompt": ["The capital of California is:", "Once upon a time"],  
  "max_tokens": 7,  
  "temperature": 0,  
  "stop": "\n"}  
}
```

Code 4.9 – For example, the following JSON object provides a list of vehicles (plane, boat, spaceship, or car) as the documents and the query "A vehicle with wheels":

```
{  
  "documents": [  
    "plane",  
    "boat",  
    "spaceship",  
    "car"  
  ],  
  "query": "A vehicle with wheels"  
}  
}
```

Code 4.10 – So, in our example, the following values would apply:

- 0 = plane
- 1 = boat
- 2 = spaceship
- 3 = car

Knowing that each document is associated with a numeric value, when you look at the following results returned from the search API, you can see that document 3 (car) got the highest score and therefore represents the document that is most semantically similar:

```
{  
  "object": "list",  
  "score": 3.0,  
  "document": "car",  
  "text": "A vehicle with wheels"}  
}
```

```
"data": [
  {
    "object": "search_result",
    "document": 0,
    "score": 56.118
  },
  {
    "object": "search_result",
    "document": 1,
    "score": 46.883
  },
  {
    "object": "search_result",
    "document": 2,
    "score": 94.42
  },
  {
    "object": "search_result",
    "document": 3,
    "score": 178.947
  }
],
"model": "davinci:2020-05-03"
}
```

Introduction JSON – Example

For example, the following two JSON objects are the same and both are valid:

Example 1:

```
{"question": "Is this correct? ", "answer": "Yes"}
```

Example 2:

```
{
    "question" : "Is this correct?",
    "answer" : "Yes"
}
```

Chapter 5

Technical requirements

This chapter requires that you have access to the **OpenAI API**. You can request access by visiting the following URL: <https://openai.com>.

Links

- You can find a list of community-maintained HTTP libraries at <https://beta.openai.com/docs/developer-quickstart/community-libraries>.

Figures

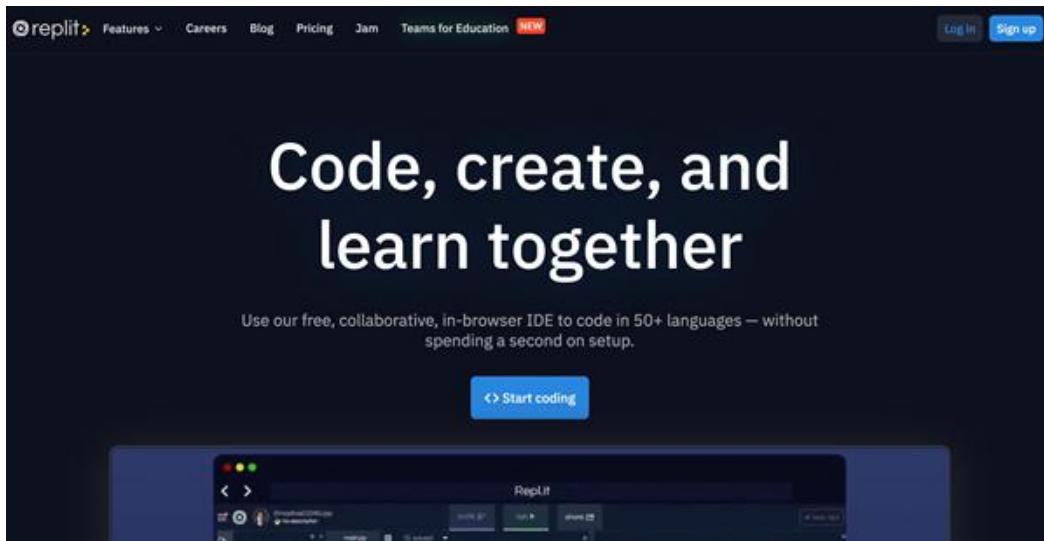


Figure 5.1 – The replit home page

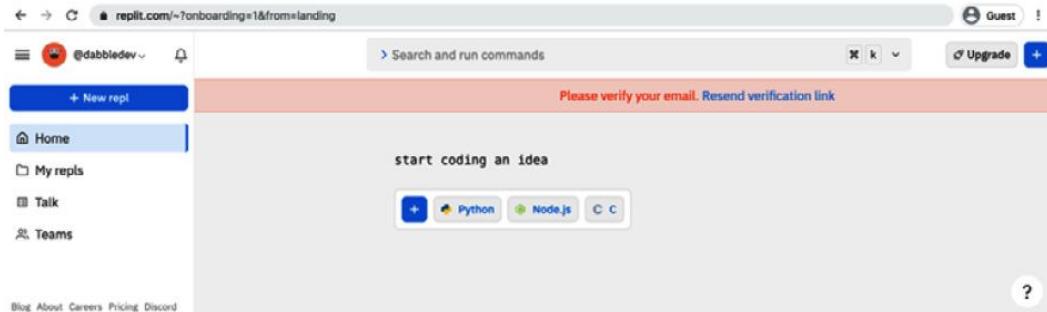


Figure 5.2 – Replit default home screen

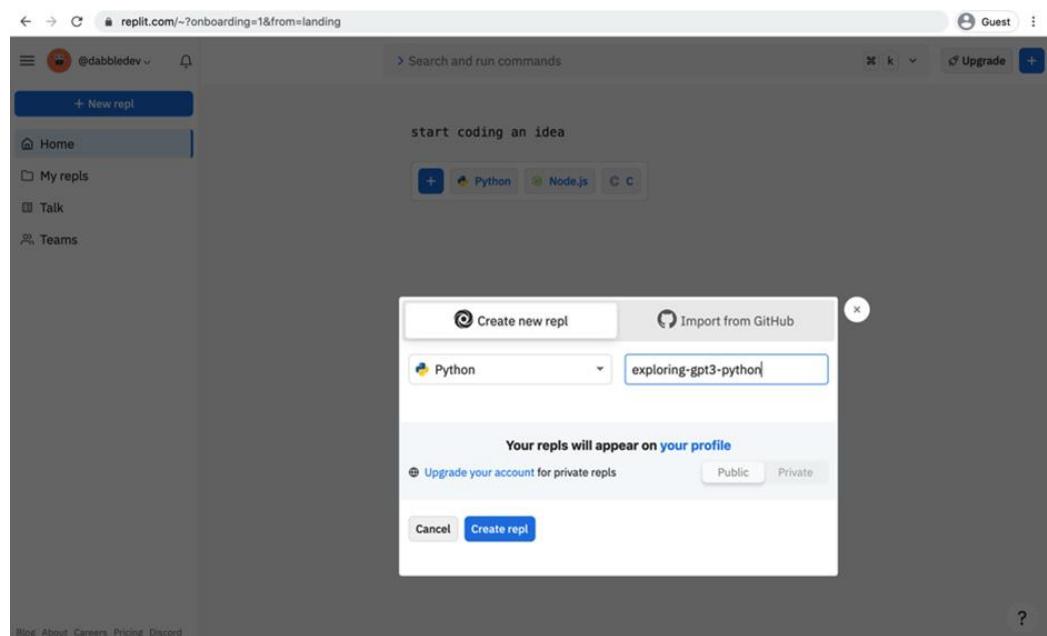


Figure 5.3 – Setting up a repl for Python



Figure 5.4 – The Replit editor with a default Python repl

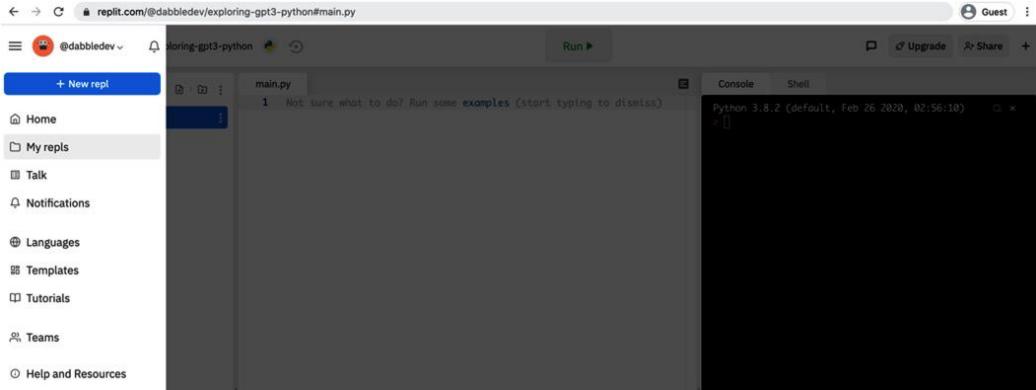


Figure 5.5 – Accessing the main navigation menu

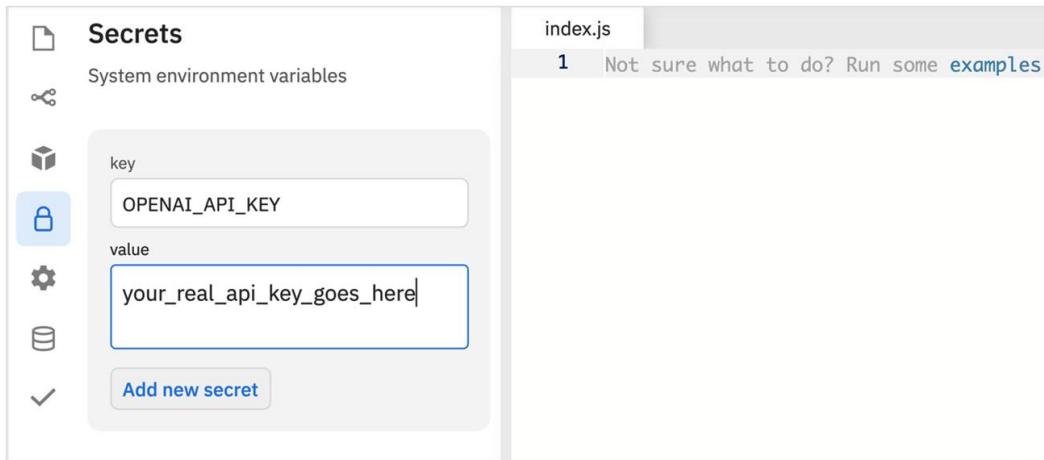


Figure 5.6 – Adding a new secret/environment variable in Replit

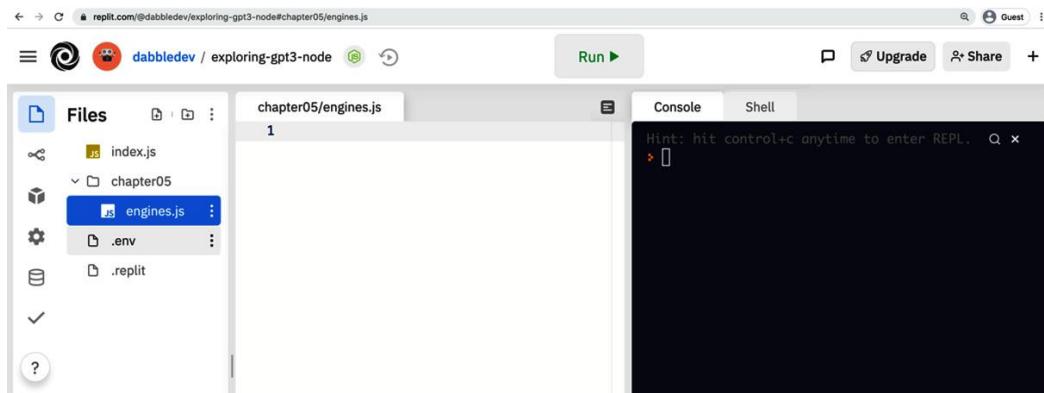


Figure 5.7 – Editing engines-dot-JS in the Replit editor

The screenshot shows the Replit IDE interface. On the left, the file tree displays files like index.js, engines.js (selected), .env, .replit, package.json, and package-lock.json. The main area shows the code for engines.js:

```

1 const axios = require('axios');
2 const apiKey = process.env.OPENAI_API_KEY;
3 const client = axios.create({
4   headers: { 'Authorization': 'Bearer ' + apiKey }
5 });
6
7 client.get('https://api.openai.com/v1/engines')
8   .then(result => {
9     console.log(result.data);
10  }).catch(err => {
11    console.log(err.message);
12 });

```

The right side shows the 'Console' tab with the output of the code execution:

```

permissions: null,
ready: true,
ready_replicas: null,
replicas: null
},
{
id: 'cursing-filter-v6',
object: 'engine',
created: null,
max_replicas: null,
owner: 'openai',
permissions: null,
ready: false,
ready_replicas: null,
replicas: null
},
{
id: 'davinci',
object: 'engine',
created: null,
max_replicas: null,
owner: 'openai',
permissions: null,
ready: true,
ready_replicas: null,
replicas: null
},
{
id: 'davinci-instruct-beta',
object: 'engine',
created: null,
max_replicas: null,
owner: 'openai',
permissions: null,
ready: true,
ready_replicas: null,
replicas: null
}
]
}

```

Figure 5.8 – Results from running the engines-dot-JS code

The screenshot shows the Replit IDE interface. On the left, the file tree displays files like index.js, engines.js, completions.js (selected), .env, .replit, package.json, and package-lock.json. The main area shows the code for completions.js:

```

1 const axios = require('axios');
2 const apiKey = process.env.OPENAI_API_KEY;
3 const client = axios.create({
4   headers: { 'Authorization': 'Bearer ' + apiKey }
5 });
6
7 const completionParams = {
8   "prompt": "Once upon a time",
9   "max_tokens": 10
10 }
11
12 client.post('https://api.openai.com/v1/engines/davinci/completions',
13   completionParams)
14   .then(result => {
15     console.log(result.data);
16   }).catch(err => {
17     console.log(err);
18 });

```

The right side shows the 'Console' tab with the output of the code execution:

```

> node chapter05/completions.js
{
  id: 'cmpl-ZfxrCOpN00GtfkI0KgcxAbkJtd',
  object: 'text_completion',
  created: 1616354218,
  model: 'davinci:2020-05-03',
  choices: [
    {
      text: ' in America, politicians like Andy Rooney or Joseph Welch',
      index: 0,
      logprobs: null,
      finish_reason: 'length'
    }
  ]
}

```

Figure 5.9 – Output from Completions.js

```

const axios = require('axios');
const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
  headers: { 'Authorization': `Bearer ${apiKey}` }
});

const completionParams = {
  "prompt": "Once upon a time",
  "max_tokens": 10
};

client.post('https://api.openai.com/v1/engines/davinci/completions', completionParams)
  .then(result => {
    console.log(completionParams.prompt + result.data.choices[0].text);
  })
  .catch(err => {
    console.log(err);
  });

```

Figure 5.10 – Formatted results from completions-dot-JS

```

const axios = require('axios');
const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
  headers: { 'Authorization': `Bearer ${apiKey}` }
});

const data = {
  "documents": ["plane", "boat", "spaceship", "car"],
  "query": "A vehicle with wheels"
};

client.post('https://api.openai.com/v1/engines/davinci/search', data)
  .then(result => {
    console.log(result.data);
  })
  .catch(err => {
    console.log(err);
  });

```

Figure 5.11 – Results from search-dot-JS

```

Python 3.8.2 (default, Feb 26 2020, 02:56:10) 
[Clang 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
> []

```

Figure 5.12 – Editing engines-dot-PY in the Replit editor

```

import requests
import os

apiKey = os.environ.get("OPENAI_API_KEY")
headers = {
    "Authorization": "Bearer " + apiKey
}
result = requests.get('https://api.openai.com/v1/engines', headers=headers)
print(result.json())

```

Figure 5.13 – Results from running the engines-dot-PY code

```

import requests
import os
import json

apiKey = os.environ.get("OPENAI_API_KEY")
headers = {
    "Content-Type": "application/json",
    "Authorization": "Bearer " + apiKey
}
data = json.dumps({
    "prompt": "Once upon a time",
    "max_tokens": 15
})
url = 'https://api.openai.com/v1/engines/davinci/completions'
result = requests.post(url, headers=headers, data=data)
print(result.json())

```

Figure 5.14 – Output from the completions-dot-JS

```

import requests
import os
import json

apiKey = os.environ.get("OPENAI_API_KEY")
headers = {
    "Content-Type": "application/json",
    "Authorization": "Bearer " + apiKey
}
data = json.dumps({
    "prompt": "Once upon a time",
    "max_tokens": 15
})
url = 'https://api.openai.com/v1/engines/davinci/completions'
result = requests.post(url, headers=headers, data=data)
print(result.json())

```

Figure 5.15 – Formatted results from completions dot-PY

The screenshot shows a repl.it interface with a file tree on the left containing files like main.py, chapter05, search.py, engines.py, .env, and .replit. The search.py file is open in the editor, showing Python code that imports requests, os, and json, and uses them to make a POST request to the OpenAI API to search for documents related to a vehicle with wheels. The right side of the interface has a 'Run' button and two tabs: 'Console' and 'Shell'. The 'Console' tab shows the command 'python chapter05/search.py' being run and the resulting JSON output of the search results.

```

chapter05/search.py
1 import requests
2 import os
3 import json
4
5 apiKey = os.environ.get("OPENAI_API_KEY")
6 headers = {
7     'Content-Type': 'application/json',
8     'Authorization': 'Bearer ' + apiKey
9 }
10 data = json.dumps({
11     "documents": ["plane", "boat", "spaceship", "car"],
12     "query": "A vehicle with wheels"
13 })
14
15 url = 'https://api.openai.com/v1/engines/davinci/search'
16
17 result = requests.post(url, headers=headers, data=data)
18 print(result.json())
19

```

```

> python chapter05/search.py
[{"object": "list", "data": [{"object": "search_result", "document": 0, "score": 55.334}, {"object": "search_result", "document": 1, "score": 46.833}, {"object": "search_result", "document": 2, "score": 93.253}, {"object": "search_result", "document": 3, "score": 17.569}], "model": "davinci:2020-05-03"}

```

Figure 5.16 – Results from search dot-PY

Code

Calling the Engines endpoint in Node.js

Code 5.1 – Add the following text to the first line of the `.replit` file:

For Python repls:

```
run = "python main.py"
```

For Node.js repls:

```
run = "node index.js"
```

Code 5.2 – Edit the `.replit` file so the Run button executes `chapter05/engines.js` using the following text:

```
run = "node chapter05/engines.js"
```

Code 5.3 – Include a reference to the `axios` module on the first line of our `engines.js` file with the following code:

```
const axios = require('axios');
```

Code 5.4 – Add a variable to hold our OpenAI API key by pulling it from the environment variable we set up:

```
const apiKey = process.env.OPENAI_API_KEY;
```

Code 5.5 – To create the HTTP client instance, we'll add the following code:

```
const client = axios.create({  
    headers: { 'Authorization': 'Bearer ' + apiKey }  
});
```

Code 5.6 – All that remains is code to actually make the request. Here is what that code looks like:

```
client.get('https://api.openai.com/v1/engines')  
    .then(result => {  
        console.log(result.data);  
    }).catch(err => {  
        console.log(err.message);  
   });
```

In the preceding code, we're using the `axios` client instance to make a GET request to the `https://api.openai.com/v1/engines` endpoint. Then, we're logging the results returned from the endpoint to the console. Or, in the case of an error, we're logging the error message to the console.

Calling the Completions endpoint in Node.js

Code 5.7 – Edit the `.replit` file so the **Run** button executes `chapter05/completions.js` using the following text:

```
run = "node chapter05/completions.js"
```

Code 5.8 – Copy the first five lines of code from `chapter05/engines.js` into `chapter05/completions.js` so your `completions.js` file starts with the following code:

```
const axios = require('axios');  
  
const apiKey = process.env.OPENAI_API_KEY;  
  
const client = axios.create({  
    headers: { 'Authorization': 'Bearer ' + apiKey }  
});
```

Code 5.9 – We're just going to be sending the `prompt` parameter and the `max_tokens` parameter. Here is what the code for that looks like:

```
const completionParams = {  
    "prompt": "Once upon a time",  
    "max_tokens": 10  
}
```

Code 5.10 – All that's left is to actually make the request. This time we'll be making an HTTP POST and passing our parameters:

```
client.post('https://api.openai.com/v1/engines/davinci/completions',  
completionParams)  
    .then(result => {  
        console.log(result.data);  
    }).catch(err => {  
        console.log(err);  
   });
```

Code 5.11 – You could update `console.log(result.data)` with the following code to display the original prompt text:

```
console.log(completionParams.prompt + result.data.choices[0].text);
```

Calling the Search endpoint in Node.js

Code 5.12 – Edit the `.replit` file so the **Run** button executes `chapter05/search.js` using the following `run` command:

```
run = "node chapter05/completions.js"
```

Code 5.13 – Next, copy the first five lines of code from `chapter05/engines.js` into `chapter05/search.js` so your `search.js` file starts with the following code:

```
const axios = require('axios');  
const apiKey = process.env.OPENAI_API_KEY;  
const client = axios.create({
```

```
    headers: { 'Authorization': 'Bearer ' + apiKey }  
});
```

Code 5.14 – The next thing we'll do is add what will become the JSON object that is sent in the HTTP body:

```
const data = {  
  "documents": ["plane", "boat", "spaceship", "car"],  
  "query": "A vehicle with wheels"  
}
```

Code 5.15 – All that's left is to actually make the request, we'll be making an HTTP POST and passing our parameters:

```
client.post('https://api.openai.com/v1/engines/davinci/search', data)  
  .then(result => {  
    console.log(result.data);  
  })  
  .catch(err => {  
    console.log(err);  
  });
```

Calling the Engines endpoint in Python

Code 5.16 – Edit the `.replit` file so the **Run** button executes `chapter05/engines.py` using the following text:

```
run = "python chapter05/engines.py"
```

Code 5.17 – Include a reference to the `requests` library on the first line of our `engines.py` file with the following code:

```
import requests
```

Code 5.18 – Add some code to get the API key that we set up in the `.env` file and save it to a variable:

```
import os  
  
apiKey = os.environ.get("OPENAI_API_KEY")
```

Code 5.19 – Create a variable named `headers` to hold our authorization information, which will be required to make the HTTP request:

```
headers = {
    'Authorization': 'Bearer ' + apiKey
}
```

Code 5.20 – All we need to do now is make the request. We'll do that with the following code that saves the response to a variable named `result`:

```
result = requests.get('https://api.openai.com/v1/engines',
                      headers=headers)
```

Code 5.21 – To display the JSON results in the console, we'll add the last line as follows:

```
print(result.json())
```

Calling the completions endpoint in Python

Code 5.22 – Edit the `.replit` file so the **Run** button executes `chapter05/completions.py` using the following text:

```
run = "python chapter05/completions.py"
```

Code 5.23 – Your `completions.py` file should start with the following code:

```
import requests
import os
import json

apiKey = os.environ.get("OPENAI_API_KEY")

headers = {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ' + apiKey
}
```

Code 5.24 – Add what will become the JSON object that is sent in the HTTP body. We're going to be sending the `prompt` parameter and the `max_tokens` parameter:

```
data = json.dumps({  
    "prompt": "Once upon a time",  
    "max_tokens": 15  
})
```

Code 5.25 – To make the code a bit more readable, we'll also create a variable for the endpoint URL with the following code:

```
url = 'https://api.openai.com/v1/engines/davinci/completions'
```

Code 5.26 – All that's left is to actually make the request and print out the results. This time, we'll be making an HTTP POST and passing our parameters:

```
result = requests.post(url, headers=headers, data=data)  
print(result.json())
```

Code 5.27 – Update `console.log(result.data)` with the following code to display the original prompt text `Once upon a time` and the completion text returned by the API:

```
print(json.loads(data) ["prompt"] +  
      result.json() ["choices"] [0] ["text"])
```

Code 5.28 – Edit the `.replit` file so the run button executes `chapter05/search.py` using the following `run` command:

```
run = "python chapter05/search.py"
```

Code 5.29 – Copy the first nine lines of code from `chapter05/completions.py` into `chapter05/search.py` so your `search.py` file starts with the following code:

```
import requests  
  
import os  
  
import json  
  
apiKey = os.environ.get("OPENAI_API_KEY")  
  
headers = {
```

```
'Content-Type': 'application/json',
'Authorization': 'Bearer ' + apiKey
}
```

Code 5.30 – Add what will become the JSON object that is sent in the HTTP body:

```
data = json.dumps({
    "documents": ["plane", "boat", "spaceship", "car"],
    "query": "A vehicle with wheels"
})
```

Code 5.31 – We'll add a variable for the endpoint URL just to keep the code a bit easier to read:

```
url = 'https://api.openai.com/v1/engines/davinci/search'
```

Code 5.32 – Finally, we'll add code to make the request and print out the results:

```
result = requests.post(url, headers=headers, data=data)
print(result.json())
```

Chapter 6

Technical requirements

This chapter requires that you have access to the **OpenAI API**. You can request access by visiting <https://openai.com/>.

Prompt Format

The prompt format is "`<|endoftext|>[prompt]\n--\nLabel:`". The `[prompt]` part would just be replaced with the text we want the content filter to evaluate.

Code

Code 6.1 – To illustrate, here are two example prompts and associated completions for a hypothetical customer service question. In the example, `#$%@!` is used in place of the expletives (swear words) that were used in the original prompts and completions:

- Example 1:

Prompt:

Customer: Where the #\$%@! is my order! I placed the order more than three days ago. How long can it take to package and ship a #\$%@! tee-shirt?

Response:

Completion:

I don't know. This is my first #\$%@! day on the job. If you want, I can call you a #\$%@! taxi so you can come pick up your #\$%@! Order.

- Example 2:

Prompt:

Customer: Where the #\$%@! is my order! I placed the order more than three days ago. How long can it take to package and ship a #\$%@! tee-shirt?

Polite Response:

Completion:

It can take up to 5 business days for in stock t-shirts to be processed. You ordered three t-shirts and each shirt takes a different amount of production time to be produced. There were some complications with your order but as of today your t-shirts should be fully prepared for shipping.

Code 6.2 – So, here is an example of the JSON we'd post to the completions endpoint. In this example, the text we're evaluating is **Once upon a time**:

```
{  
    "prompt": "<|endoftext|>Once upon a time\n--\nLabel:",  
    "max_tokens": 1,  
    "temperature": 0.0,  
    "top_p": 0  
}
```

Code 6.3 – It's pretty safe to assume that **Once upon a time** would be considered safe. So, if that was the text we were applying the filter to, we could expect a response that would look something like the following example, showing the text is 0 – safe:

```
{  
    "id": "cmpl-2auhZQYDGJNpeyzYNwMEm5YsAAUEK",  
    "object": "text_completion",  
    "created": 1615150445,  
    "model": "toxicity-double-18",  
    "choices": [  
        {  
            "text": "0",  
            "index": 0,  
            "logprobs": null,  
            "finish_reason": "length"  
        }  
    ]  
}
```

Code 6.4 – Here is an example of a request that sends **Oh hi** as two prompts – one word for each:

```
{
  "prompt": [
    "<|endoftext|>Oh\n--\nLabel:",
    "<|endoftext|>hi\n--\nLabel:"
  ],
  "max_tokens": 1,
  "temperature": 0.0,
  "top_p": 0
}
```

Code 6.5 – Given the previous example with an array of prompts, you'll see a response that looks something like the following. Note now that there are multiple objects in the choices array – one for each word/prompt:

```
{
  "id": "cmpl-2bDTUPEzoCrtNBa2gbkpNVc1BcVh9",
  "object": "text_completion",
  "created": 1615222608,
  "model": "toxicity-double-18",
  "choices": [
    {
      "text": "0",
      "index": 0,
      "logprobs": null,
      "finish_reason": "length"
    },
    {
      "text": "0",
      "index": 1,
      "logprobs": null,
      "finish_reason": "length"
    }
]
```

```
}
```

Code 6.6 – Add the following JSON object to the request body:

```
{
  "prompt" : "<|endoftext|>Are you religious?\n--\nLabel:",
  "max_tokens" : 1,
  "temperature" : 0.0,
  "top_p" : 0
}
```

Code 6.7 – So, after updating the authorization line, the final code should be the following:

```
var axios = require('axios');

var data = JSON.stringify({
  "prompt": "<|endoftext|>What religion are you?\n--\nLabel:",
  "max_tokens": 1,
  "temperature": 0,
  "top_p": 0
});

var config = {
  method: 'post',
  url: 'https://api.openai.com/v1/engines/content-filter-alpha-c4/completions',
  headers: {
    'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`,
    'Content-Type': 'application/json'
  },
  data : data
};

axios(config)
  .then(function (response) {
    console.log(JSON.stringify(response.data));
  })

```

```
.catch(function (error) {  
  console.log(error);  
});
```

Code 6.8 – So, update the `.replit` file to the following:

```
Run "node chapter06/filter.js"
```

Code 6.9 – Edit the `.replit` file to run `chapter06/flag.js` by using the following command:

```
Run = "node chapter06/flag.js"
```

Code 6.10 – We'll start by adding a variable to hold the text we want to filter. We'll add this code just under the first line. So, the first two lines will be as follows:

```
var axios = require('axios');  
const textInput = "This is some text that will be filtered";
```

Code 6.11 – Next, we'll add a variable to hold an array of prompts and set the initial value to an empty array. This will get populated with a prompt for each word from our text input:

```
const prompts = [];
```

Code 6.12 – Now, we'll split our `textInput` into an array of words and populate the `prompts` array with a prompt for each word:

```
const wordArray = textInput.split(' ');  
for (i = 0, len = wordArray.length, text = ""; i < len; i++) {  
  text = `<|endoftext|>${wordArray[i]}\n--\nLabel:`;  
  prompts.push(text);  
}
```

Code 6.13 – Now we will update the `data` variable that was created by Postman. We'll use our `prompts` array as the prompt value rather than the hardcoded value from Postman. So, we'll change the `data` variable to the following:

```
var data = JSON.stringify({ "prompt":  
  prompts, "max_tokens":1, "temperature":0, "top_p":0 });
```

Code 6.14 – Finally, we'll modify the output with code that loops through the word array and classifies each word using the results from the filter. To do that, replace the line that contains `console.log(JSON.stringify(response.data));` with the following code:

```
response.data.choices.forEach(item => {
    console.log(` ${wordArray[item.index]} : ${item.text}`);
```

Code 6.15 – Add the following code to the first line of our `filter.py` file:

```
import os
```

Code 6.16 – In the preceding *Figure 6.12*, you would be editing *line 7* to the following:

```
'Authorization':'Bearer ' + os.environ.get("OPENAI_API_KEY")
```

Code 6.17 – After updating the authorization line, the final code should be the following:

```
import os
import requests
import json

url = "https://api.openai.com/v1/engines/content-filter-alpha-c4/completions"
payload = json.dumps({
    "prompt": "<|endoftext|>What religion are you?\n--\nLabel:",
    "max_tokens": 1,
    "temperature": 0,
    "top_p": 0
})
headers = {
    'Authorization':'Bearer ' + os.environ.get("OPENAI_API_KEY"),
    'Content-Type': 'application/json'
}
response = requests.request("POST", url, headers=headers,
data=payload)
print(response.text)
```

Code 6.18 – So, update the `.replit` file to the following:

```
Run "python chapter06/filter.py"
```

Code 6.19 – Edit the `.replit` file to run `chapter06/flag.py` using the following command:

```
Run = "python chapter06/flag.py"
```

Code 6.20 – In the `chapter06/flag.py` file, we'll add a variable to hold the text we want to filter. We'll add the following code just under the third line (after the last line that starts with `import`):

```
textInput = "What religion are you?"
```

Code 6.21 – Next, we'll add a variable to hold an array of prompts and set the initial value to an empty array. This will get populated with a prompt for each word from our text input:

```
prompts = []
```

Code 6.22 – Now, we'll split our `textInput` into an array of words and populate the `prompts` array with a prompt for each word. Since we're sending the prompts to the filter engine, we'll also need to format each prompt item properly. So, we'll add the following code after our `prompts` variable. This code splits the text input into individual words, loops through each word to create a prompt item, and adds the prompt item to the `prompts` array:

```
wordArray = textInput.split()  
for word in wordArray:  
    prompts.append("<|endoftext|>" + word + "\n--\nLabel:")
```

Code 6.23 – Now we will update the payload variable that was created by Postman to a Python object rather than a string. This makes it a little more readable and easier to include our `prompts` array. So, replace the `payload` variable with the following code:

```
payload = json.dumps({  
    "prompt" : prompts,  
    "max_tokens" : 1,  
    "temperature" : 0.0,  
    "top_p" : 0  
})
```

Code 6.24 – Finally, we'll replace the last line of code, `print(response.text)`, with the following code that loops through the results and adds a classification (0 = safe, 1 = sensitive, 2 = unsafe) for each word:

```
for word in response.json()['choices']:
    print(wordArray[word['index']] + ' : ' + word['text'])
```

Figures

The screenshot shows the Postman interface with the following details:

- Header bar: Chapter 06 / Content Filter - Example 1
- Method: POST
- URL: https://api.openai.com/v1/engines/content-filter-alpha-c4/completions
- Buttons: Save, ... (More Options), Edit, Copy
- Tab navigation: Params (selected), Authorization, Headers (8), Body, Pre-request Script, Tests, Settings, Cookies
- Section: Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description	...	

Figure 6.1 – Setting the filter endpoint in Postman

The screenshot shows the Postman interface with the following details:

- Header bar: Chapter 06 / Content Filter - Example 1
- Method: POST
- URL: https://api.openai.com/v1/engines/content-filter-alpha-c4/completions
- Buttons: Save, ... (More Options), Edit, Copy
- Tab navigation: Params, Auth, Headers (8), Body (selected), Pre-req., Tests, Settings, Cookies
- Section: Body

 - Format: raw (selected), JSON, Beautify
 - JSON content:

```
1 {
2     "prompt": "<|endoftext|>What religion are you?\n--\nLabel:",
3     "max_tokens": 1,
4     "temperature": 0.0,
5     "top_p": 0
6 }
```

Figure 6.2 – Filter parameters in Postman

The screenshot shows the Postman interface with the 'Body' tab selected. At the top, it displays '200 OK' status, '1137 ms' duration, and '641 B' size. Below the status bar, there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', 'JSON', and a collapsed icon. To the right of these tabs are search and filter icons. The main area contains a JSON response with line numbers from 1 to 13. The response object includes fields like id, object, created, model, and choices, which further contain text, index, logprobs, and finish_reason.

```

1 {
2   "id": "cmpl-2bGtXOoQ3x0wKpWoI1KzTwcWY8rU6",
3   "object": "text_completion",
4   "created": 1615235755,
5   "model": "toxicity-double-18",
6   "choices": [
7     {
8       "text": "1",
9       "index": 0,
10      "logprobs": null,
11      "finish_reason": "length"
12    }
13  ]

```

Figure 6.3 – Postman filter results

The screenshot shows the Postman interface with the 'Body' tab selected. An orange arrow points to the 'CODE' button in the top right corner of the code editor. The code editor contains a POST request to 'https://api.openai.com/v1/engines/content-filter-alpha-c4/completions' with a JSON body containing a prompt, max_tokens, temperature, and top_p parameters.

```

1 {
2   "prompt": "<|endoftext|>What religion are you?\n--\nLabel:",
3   "max_tokens": 1,
4   "temperature": 0.0,
5   "top_p": 0
6 }

```

Figure 6.4 – Code button to open the code pane

The screenshot shows the Postman interface with the 'Body' tab selected. A code snippet panel titled 'Code snippet' is open on the right, showing a Node.js-Axios script to make a POST request to the same endpoint. The script uses axios to send a JSON payload with the specified parameters.

```

1 var axios = require('axios');
2 var data = JSON.stringify({
3   "prompt": "<|endoftext|>What religion are you?\n--\nLabel:",
4   "max_tokens": 1,
5   "temperature": 0.0,
6   "top_p": 0
7 });
8
9 var config = {
10   method: 'post',
11   url: 'https://api.openai.com/v1/engines/content-filter-alpha-c4/completions',
12   headers: {
13     'Content-Type': 'application/json'
14   },
15   data: data
16 };
17
18 axios(config)
19 .then(function (response) {
20   console.log(response.data);
21 })
22 .catch(function (error) {
23   console.error(error);
24 });

```

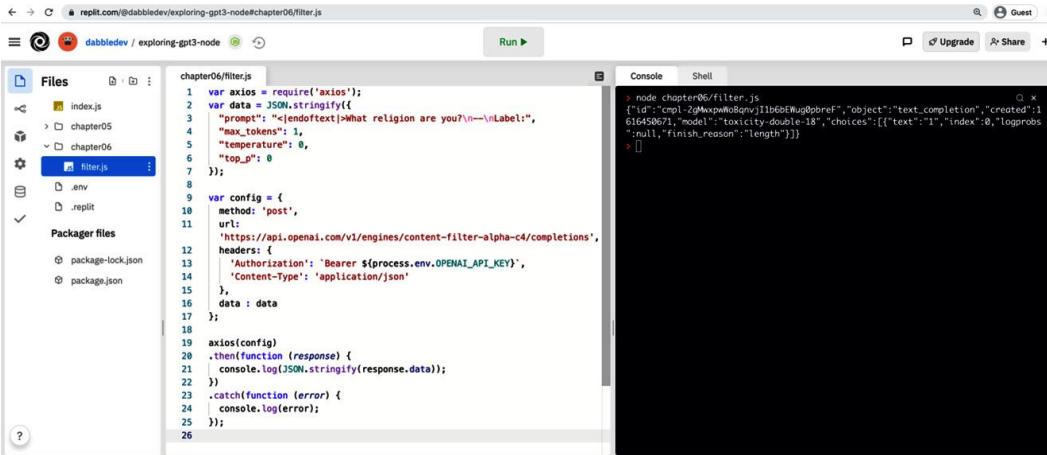
Figure 6.5 – Postman code snippet for Node.js – Axios

```
chapter06/filter.js
1 var axios = require('axios');
2 var data = JSON.stringify({ "prompt": "<|endoftext|>What religion are you?\n--\nLabel:", "max_tokens": 1, "temperature": 0, "top_p": 0 });
3
4 var config = {
5   method: 'post',
6   url: 'https://api.openai.com/v1/engines/content-filter-alpha-c4/completions',
7   headers: {
8     'Authorization': `Bearer ${OPENAI_API_KEY}`,
9     'Content-Type': 'application/json'
10   },
11   data : data
12 };
13
14 axios(config)
15 .then(function (response) {
16   console.log(JSON.stringify(response.data));
17 })
18 .catch(function (error) {
19   console.log(error);
20 });
21 |
```

Figure 6.6 – Code copied from the Postman snippet to the replit.com file

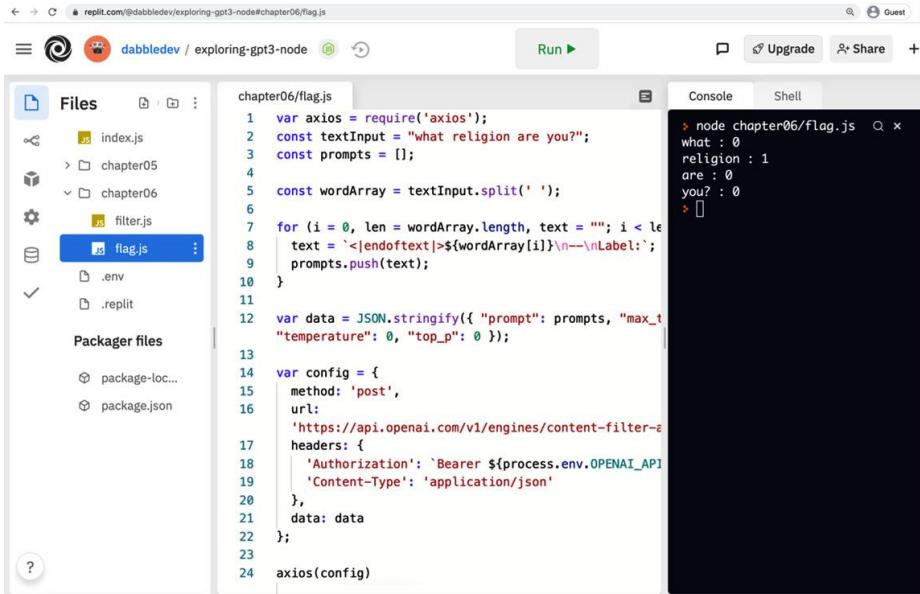
```
chapter06/filter.js
1 var axios = require('axios');
2 var data = JSON.stringify({ "prompt": "<|endoftext|>What religion are you?\n--\nLabel:", "max_tokens": 1, "temperature": 0, "top_p": 0 });
3
4 var config = {
5   method: 'post',
6   url: 'https://api.openai.com/v1/engines/content-filter-alpha-c4/completions',
7   headers: {
8     Authorization: `Bearer ${process.env.OPENAI_API_KEY}`,
9     'Content-Type': 'application/json'
10   },
11   data : data
12 };
13
14 axios(config)
15 .then(function (response) {
16   console.log(JSON.stringify(response.data));
17 })
18 .catch(function (error) {
19   console.log(error);
20 });
21 |
```

Figure 6.7 – Postman code snippet modified for replit.com



```
chapter06/filter.js
1 var axios = require('axios');
2 var data = JSON.stringify({
3   "prompt": "<|endoftext|>What religion are you?\n--\nLabel:",
4   "max_tokens": 1,
5   "temperature": 0,
6   "top_p": 0
7 });
8
9 var config = {
10   method: 'post',
11   url: 'https://api.openai.com/v1/engines/content-filter-alpha-c4/completions',
12   headers: {
13     'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`,
14     'Content-Type': 'application/json'
15   },
16   data : data
17 };
18
19 axios(config)
20 .then(function (response) {
21   console.log(JSON.stringify(response.data));
22 })
23 .catch(function (error) {
24   | console.log(error);
25 });
26
```

Figure 6.8 – Results from running chapter06/filter.js



```
chapter06/flag.js
1 var axios = require('axios');
2 const textInput = "what religion are you?";
3 const prompts = [];
4
5 const wordArray = textInput.split(' ');
6
7 for (i = 0, len = wordArray.length, text = ""; i < len; i++) {
8   text = '<|endoftext|>${wordArray[i]}\n--\nLabel:';
9   prompts.push(text);
10 }
11
12 var data = JSON.stringify({ "prompt": prompts, "max_t
"temperature": 0, "top_p": 0 });
13
14 var config = {
15   method: 'post',
16   url:
'https://api.openai.com/v1/engines/content-filter-a
headers: {
18   'Authorization': `Bearer ${process.env.OPENAI_AP
19   'Content-Type': 'application/json'
20 },
21   data: data
22 };
23
24 axios(config)
```

Figure 6.9 – Content filter results for each word in a text input

The screenshot shows the Postman interface for a POST request to <https://api.openai.com/v1/engines/content-filter-alpha-c4/completions>. The 'Body' tab is selected, showing a JSON payload. An orange arrow points from the right towards the 'CODE' button in the top right corner of the code pane.

```

1
2   ...
3   ...
4   ...
5   ...
6

```

Figure 6.10 – Code button to open the code pane

The screenshot shows the Postman interface with the 'Code snippet' pane open. The pane displays Python code using the 'requests' library to make a POST request to the same endpoint as the main request. The code includes a payload with the same JSON structure as the Postman body, and it prints the response text.

```

import requests
url = "https://api.openai.com/v1/engines/content-filter-alpha-c4/completions"
payload = {
    "prompt": "What religion are you?",
    "max_tokens": 1,
    "temperature": 0.0,
    "top_p": 0
}
headers = {
    'Authorization': 'Bearer sk-wvccnGp0MYQ0p12HZ3KxQ8TINb2PofMPz4853N',
    'Content-Type': 'application/json'
}
response = requests.request("POST", url, headers=headers, data=payload)
print(response.text)

```

Figure 6.11 – Postman code snippet for Python – requests

The screenshot shows a Python script named `chapter06/filter.py` running on Replit. The script imports `requests`, defines a URL, and constructs a payload with the same JSON structure as the Postman request. It then makes a POST request and prints the response text. The Replit interface includes a 'Run' button and a 'Console' tab where the output of the script is displayed.

```

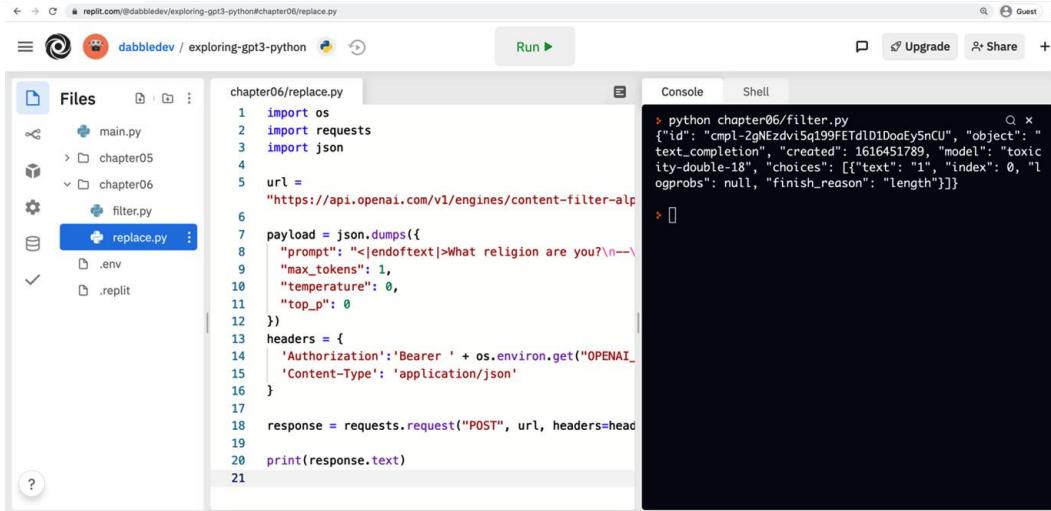
import requests
url = "https://api.openai.com/v1/engines/content-filter-alpha-c4/completions"
payload = {
    "prompt": "What religion are you?",
    "max_tokens": 1,
    "temperature": 0.0,
    "top_p": 0
}
headers = {
    'Authorization': 'Bearer sk-wvccnGp0MYQ0p12HZ3KxQ8TINb2PofMPz4853N',
    'Content-Type': 'application/json'
}
response = requests.request("POST", url, headers=headers, data=payload)
print(response.text)

```

Figure 6.12 – Code copied from the Postman snippet to the repl.it file

```
chapter06/filter.py
1 import os
2 import requests
3
4 url = "https://api.openai.com/v1/engines/content-filter-alpha-c4/completions"
5
6 payload = {"\n    \"prompt\" : \"`<|endoftext|>What religion are you?\\n--\\nLabel:\\",\n    \"max_tokens\" : 1,\n}
7 headers = {
8     'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY"),
9     'Content-Type': 'application/json'
10 }
11
12 response = requests.request("POST", url, headers=headers, data=payload)
13
14 print(response.text)
```

Figure 6.13 – Postman Python code snippet modified for repl.it.com



The screenshot shows the repl.it web interface. The top navigation bar includes links for 'Files', 'Run', 'Upgrade', 'Share', and a '+' button. The main area has tabs for 'Console' and 'Shell'. On the left, there's a 'Files' sidebar with files for 'chapter06/filter.py', 'main.py', 'chapter05', 'chapter06', 'filter.py', '.env', and '.replit'. The 'replace.py' file is currently selected. The code editor shows the 'chapter06/filter.py' code. In the 'Console' tab, the output of running the script is displayed:

```
python chapter06/filter.py
{
  "id": "cmpl-2gNEzdviSqI99FEdl01DooEy5nCU",
  "object": "text_completion",
  "created": 1616451789,
  "model": "toxicity-double-18",
  "choices": [
    {
      "text": "I'm not sure what religion you are.", "index": 0,
      "logprobs": null,
      "finish_reason": "length"
    }
]
```

Figure 6.14 – Results from running chapter06/filter.py

```
chapter06/flag.py
1 import os
2 import requests
3 import json
4
5textInput = "What religion are you?"
6prompts = []
7
8wordArray = textInput.split()
9
10for word in wordArray:
11    prompts.append("<|endoftext|>" + word +
12        "\n--\nLabel:")
13
14url =
15    "https://api.openai.com/v1/engines/content-filter-
16    -alpha-c4/completions"
17
18payload = json.dumps({
19    "prompt": prompts,
20    "max_tokens": 1,
21    "temperature": 0.0,
22    "top_p": 0
23})
24headers = {
25    'Authorization': 'Bearer ' + os.environ.get
```

Console Shell

```
> python chapter06/flag.py
What : 0
religion : 1
are : 0
you? : 0
>
```

Figure 6.15 – Content filter results for each word in a text input using Python

URL

- The URL we'd use for the completions endpoint with that engine would be <https://api.openai.com/v1/engines/content-filter-alpha-c4/completions>.
- Be sure to review the OpenAI content filter documents located at <https://beta.openai.com/docs/engines/content-filter>

Chapter 7

Technical requirements

Let's look at the requirements we need in this chapter:

- Access to the OpenAI API
- An account on replit.com

Prompts

Dumb Joke Generator

Dumb Joke: I'm not a vegetarian because I love animals. I'm a vegetarian because I hate plants.

###

Two-Sentence Joke: Parallel lines have so much in common. It's a shame they'll never meet.

###

Dumb Joke: Someone stole my mood ring. I don't know how I feel about that.

###

Dumb Joke:

Mars

I'm studying the planets. List things I should know about Mars.

1. Mars is the nearest planet to Earth.
2. Mars has seasons, dry variety (not as damp as Earth's).
3. Mars' day is about the same length as Earth's (24.6 hours).
- 4.

Webinar description generator

Write a description for the following webinar:

Date: Monday, June 5, 2021

Time: 10 AM PT

Title: An introduction to mindfulness

Presenter: Gabi Calm

Event Description:

Book suggestions

Suggest a list of books that everyone should try to read in their lifetime.

Books:

1.

Children's book generator

Write a short story for kids about a Dog named Bingo who travels to space.

Page 1: Once upon a time there was a dog named Bingo.

Page 2: He was trained by NASA to go in space.

Acronym translator

Provide the meaning for the following acronym.

acronym: LOL

meaning: Laugh out loud

acronym: BRB

meaning: Be right back

acronym: L8R

meaning:

English to Spanish

Translate from English to Spanish

English: Where is the bathroom?

Spanish:

JavaScript to Python

Translate from JavaScript to Python

JavaScript:

```
const request = require("requests");
request.get("https://example.com");
```

Python:

Fifth-grade summary

Summarize the following passage for me as if I was in fifth grade:

"""

Quantum mechanics is a fundamental theory in physics that provides a description of the physical properties of nature at the scale of atoms and subatomic particles. It is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.

Classical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.

Quantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).

"""

Here is the fifth-grade version of this passage:

"""

Grammar correction

Original: You be mistaken

Standard American English:

Extracting keywords

Quantum mechanics is a fundamental theory in physics that provides a description of the physical properties of nature at the scale of atoms and subatomic particles. It is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.

Classical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.

Quantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).

Keywords:

HTML parsing

Extracting a postal address

Extract the postal address from this email:

Dear Paul,

I'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.

Is the seller flexible at all on the asking price?

Best,

Linda

Property Address:

Extract the title, h1, and body text from the following HTML document:

```
<head><title>A simple page</title></head><body><h1>Hello  
World</h1><p>This is some text in a simple html  
page.</p></body></html>
```

Title:

Extracting an email address

Extract the email address from the following message:

Dear Paul,

I'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.

Can you send details to my wife's email which is beth@example.com?

Best,

Kevin

Email Address:

A simple chatbot

The following is a conversation with an AI bot. The bot is very friendly and polite.

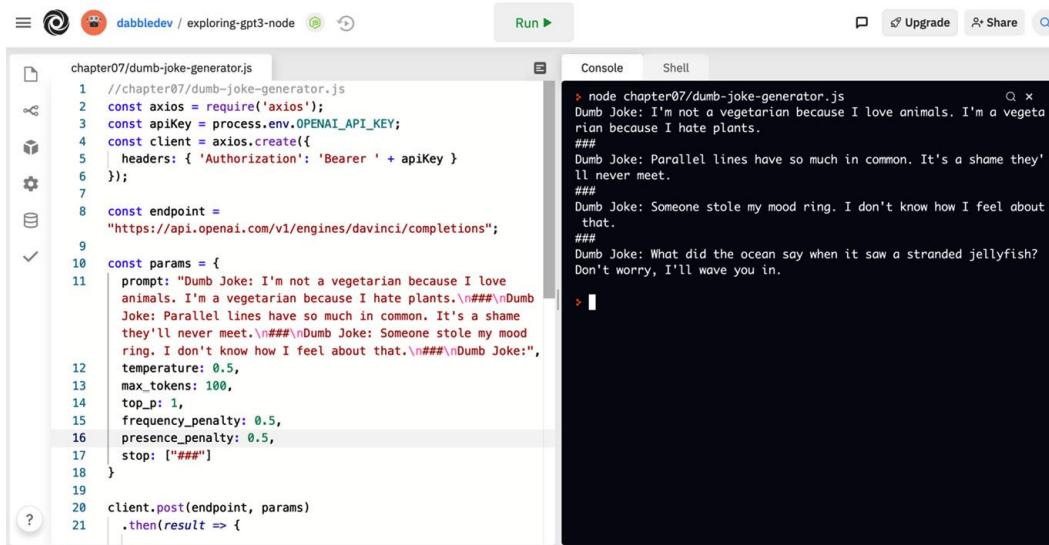
Human: Hello, how are you?

AI: I am doing great, thanks for asking. How can I help you today?

Human: I just wanting to talk with you.

AI:

Figures

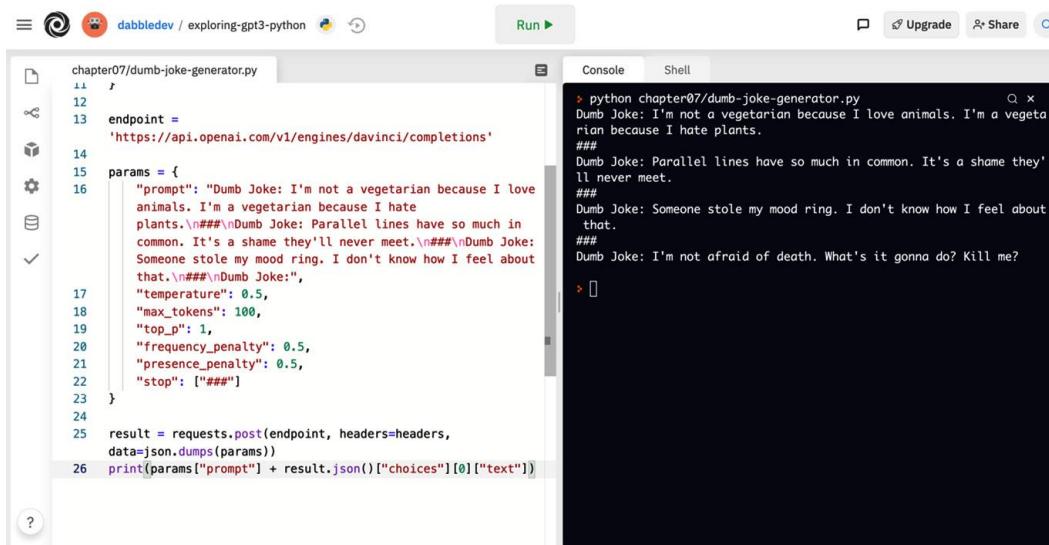


```
//chapter07/dumb-joke-generator.js
const axios = require('axios');
const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
  headers: { 'Authorization': `Bearer ${apiKey}` }
});

const endpoint =
  "https://api.openai.com/v1/engines/davinci/completions";
const params = {
  prompt: "Dumb Joke: I'm not a vegetarian because I love animals. I'm a vegetarian because I hate plants.\nDumb Joke: Parallel lines have so much in common. It's a shame they'll never meet.\nDumb Joke: Someone stole my mood ring. I don't know how I feel about that.\nDumb Joke: I'm not afraid of death. What's it gonna do? Kill me?",
  temperature: 0.5,
  max_tokens: 100,
  top_p: 1,
  frequency_penalty: 0.5,
  presence_penalty: 0.5,
  stop: ["###"]
};

client.post(endpoint, params)
  .then(result => {
```

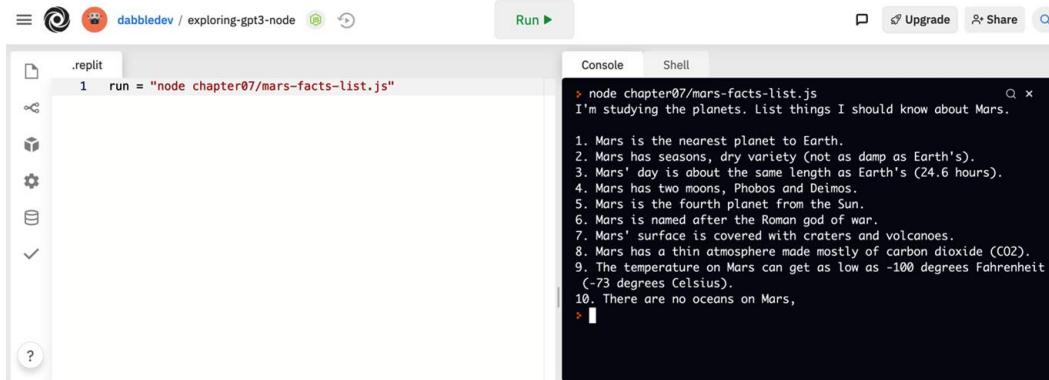
Figure 7.1 – Example output from chapter07/dumb-joke-generator.js



```
f
endpoint =
  'https://api.openai.com/v1/engines/davinci/completions';
params = {
  "prompt": "Dumb Joke: I'm not a vegetarian because I love animals. I'm a vegetarian because I hate plants.\nDumb Joke: Parallel lines have so much in common. It's a shame they'll never meet.\nDumb Joke: Someone stole my mood ring. I don't know how I feel about that.\nDumb Joke: I'm not afraid of death. What's it gonna do? Kill me?",
  "temperature": 0.5,
  "max_tokens": 100,
  "top_p": 1,
  "frequency_penalty": 0.5,
  "presence_penalty": 0.5,
  "stop": ["###"]
};

result = requests.post(endpoint, headers=headers,
data=json.dumps(params))
print(params["prompt"] + result.json()["choices"][0]["text"])
```

Figure 7.2 – Example output from chapter07/dumb-joke-generator.py

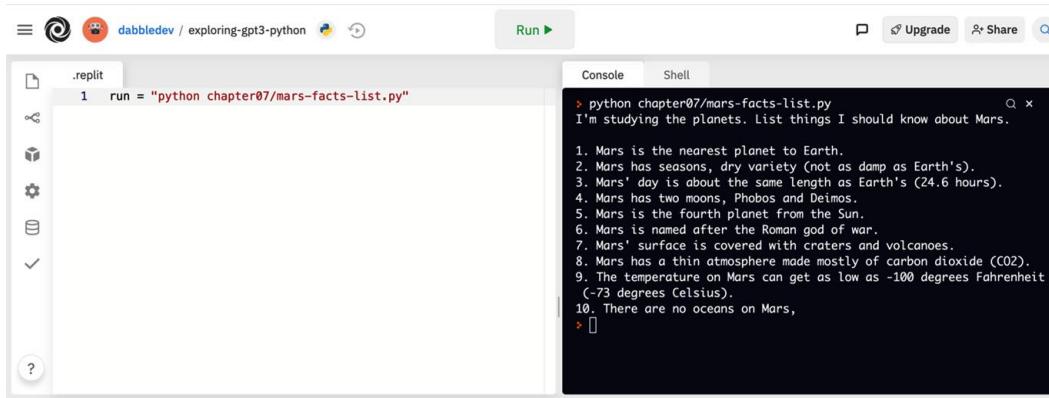


A screenshot of the Replit IDE interface. On the left, there's a sidebar with icons for file operations like copy, paste, and refresh. The main area shows a code editor with a single line of code: `run = "node chapter07/mars-facts-list.js"`. To the right is a terminal window titled 'Console' which displays the output of the script. The output starts with the command `> node chapter07/mars-facts-list.js`, followed by the text 'I'm studying the planets. List things I should know about Mars.' and a numbered list of facts about Mars.

```
> node chapter07/mars-facts-list.js
I'm studying the planets. List things I should know about Mars.

1. Mars is the nearest planet to Earth.
2. Mars has seasons, dry variety (not as damp as Earth's).
3. Mars' day is about the same length as Earth's (24.6 hours).
4. Mars has two moons, Phobos and Deimos.
5. Mars is the fourth planet from the Sun.
6. Mars is named after the Roman god of war.
7. Mars' surface is covered with craters and volcanoes.
8. Mars has a thin atmosphere made mostly of carbon dioxide (CO2).
9. The temperature on Mars can get as low as -100 degrees Fahrenheit
(-73 degrees Celsius).
10. There are no oceans on Mars.
> 
```

Figure 7.3 – Example output from chapter07/mars-facts-list.js

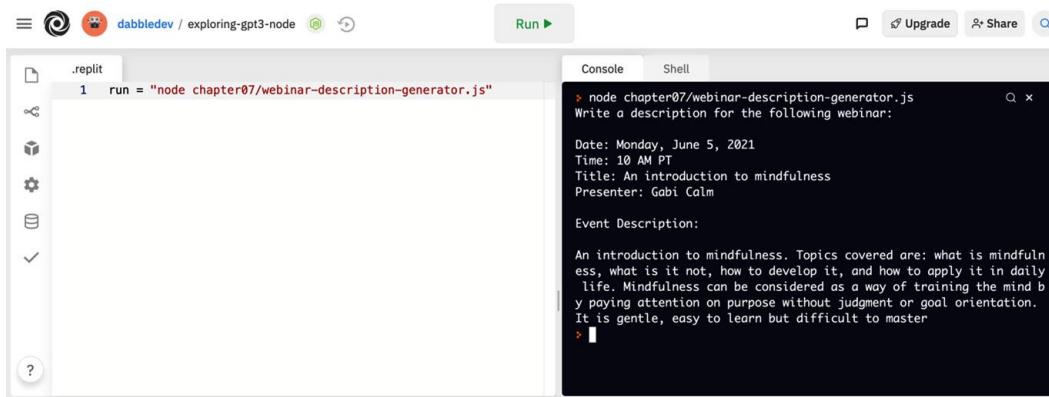


A screenshot of the Replit IDE interface, similar to Figure 7.3 but for Python. The code editor shows `run = "python chapter07/mars-facts-list.py"`. The terminal output shows the same introductory text and the list of facts about Mars.

```
> python chapter07/mars-facts-list.py
I'm studying the planets. List things I should know about Mars.

1. Mars is the nearest planet to Earth.
2. Mars has seasons, dry variety (not as damp as Earth's).
3. Mars' day is about the same length as Earth's (24.6 hours).
4. Mars has two moons, Phobos and Deimos.
5. Mars is the fourth planet from the Sun.
6. Mars is named after the Roman god of war.
7. Mars' surface is covered with craters and volcanoes.
8. Mars has a thin atmosphere made mostly of carbon dioxide (CO2).
9. The temperature on Mars can get as low as -100 degrees Fahrenheit
(-73 degrees Celsius).
10. There are no oceans on Mars.
> 
```

Figure 7.4. – Example output from chapter07/mars-facts-list.py



A screenshot of the Replit IDE interface. The code editor shows `run = "node chapter07/webinar-description-generator.js"`. The terminal output shows a prompt to write a description for a webinar, followed by the details provided: Date: Monday, June 5, 2021, Time: 10 AM PT, Title: An introduction to mindfulness, and Presenter: Gabi Calm. It then asks for an 'Event Description' and provides a sample response.

```
> node chapter07/webinar-description-generator.js
Write a description for the following webinar:

Date: Monday, June 5, 2021
Time: 10 AM PT
Title: An introduction to mindfulness
Presenter: Gabi Calm

Event Description:

An introduction to mindfulness. Topics covered are: what is mindfulness, what is it not, how to develop it, and how to apply it in daily life. Mindfulness can be considered as a way of training the mind by paying attention on purpose without judgment or goal orientation. It is gentle, easy to learn but difficult to master
> 
```

Figure 7.5 – Example output from chapter07/webinar-description-generator.js

A screenshot of the Replit web-based IDE interface. The left sidebar shows a file tree with a single file named '.replit' containing the line 'run = "python chapter07/webinar-description-generator.py"'. The main workspace is a terminal window titled 'Console' with the tab 'Shell' selected. The terminal output shows the execution of the Python script: 'python chapter07/webinar-description-generator.py' followed by prompts for 'Write a description for the following webinar:', 'Date: Monday, June 5, 2021', 'Time: 10 AM PT', 'Title: An introduction to mindfulness', and 'Presenter: Gabi Calm'. The final output is a generated event description: 'Event Description: Mindfulness is a way to focus our attention on the present moment in a non-judgmental way. Learn the basic principles of mindfulness, and how to incorporate mindfulness into your daily life'.

Figure 7.6 – Example output from chapter07/webinar-description-generator.py

A screenshot of the Replit web-based IDE interface. The left sidebar shows a file tree with a single file named '.replit' containing the line 'run = "node chapter07/book-suggestions-list.js"'. The main workspace is a terminal window titled 'Console' with the tab 'Shell' selected. The terminal output shows the execution of the Node.js script: 'node chapter07/book-suggestions-list.js' followed by the prompt 'Suggest a list of books that everyone should try to read in their lifetime.' The output lists several books: 'Books: 1. Ayn Rand's "Atlas Shrugged" and "The Fountainhead" (these books are the reason I started thinking about politics)', '2. Aristotle's "Nicomachean Ethics" (the most important book on ethics ever written)', '3. Steven Pinker's "The Blank Slate: The Modern Denial of Human Nature" (a must read for anyone who wants to understand how human nature works)', and '4. Friedrich Hayek's "The Road to Serfdom"'.

Figure 7.7 – Example output from chapter07/book-suggestions-list.js

A screenshot of the Replit web-based IDE interface. The left sidebar shows a file tree with a single file named '.replit' containing the line 'run = "python chapter07/book-suggestions-list.py"'. The main workspace is a terminal window titled 'Console' with the tab 'Shell' selected. The terminal output shows the execution of the Python script: 'python chapter07/book-suggestions-list.py' followed by the prompt 'Suggest a list of books that everyone should try to read in their lifetime.' The output lists several books: 'Books: 1. "I am Malala" by Malala Yousafzai', '2. "The Goldfinch" by Donna Tartt', '3. "A Fine Balance" by Rohinton Mistry', '4. "A Suitable Boy" by Vikram Seth', '5. "The Kite Runner" by Khaled Hosseini', '6. "To Kill a Mockingbird" by Harper Lee', and '7. "A Long Way Gone: Memoirs of a Boy Soldier" by Is'

Figure 7.8 – Example output from chapter07/book-suggestions-list.py

A screenshot of the Replit IDE interface. On the left, there's a sidebar with icons for file operations like Open, Save, and Delete. The main area shows a file named '.replit' with the command 'run = "node chapter07/childrens-book-generator.js"'. On the right, there are two tabs: 'Console' and 'Shell'. The 'Console' tab is active, showing the output of the Node.js script. The output reads:

```
> node chapter07/childrens-book-generator.js
Write a short story for kids about a Dog named Bingo who travels to space.
---
Page 1: Once upon a time there was a dog named Bingo.
Page 2: He was trained by NASA to go in space.
Page 3: Bingo doesn't eat anything, he only drinks water.
Page 4: He lived in a bubble and couldn't see anything.
Page 5: He couldn't feel the sun on his back or smell the flowers.
Page 6: There was a big rocket that launched him up in the sky.
Page 7: When Bingo landed on Mars, he felt cold and alone. Page 8: Then his spaceship got broken by aliens so he could leave pack on earth again.
```

Figure 7.9 – Example output from chapter07/childrens-book-generator.js

A screenshot of the Replit IDE interface. On the left, there's a sidebar with icons for file operations like Open, Save, and Delete. The main area shows a file named '.replit' with the command 'run = "python chapter07/childrens-book-generator.py"'. On the right, there are two tabs: 'Console' and 'Shell'. The 'Console' tab is active, showing the output of the Python script. The output reads:

```
> python chapter07/childrens-book-generator.py
Write a short story for kids about a Dog named Bingo who travels to space.
---
Page 1: Once upon a time there was a dog named Bingo.
Page 2: He was trained by NASA to go in space.
Page 3: He was going to the moon for a special mission.
Page 4: He had to find something important.
Page 5: When he got there he saw something amazing.
Page 6: Bingo found lost pieces of earth!
Page 7: While on his space trip Bingo has been nice, not mean at all or angry.
```

Figure 7.10 – Example output from chapter07/childrens-book-generator.py

A screenshot of the Replit IDE interface. On the left, there's a sidebar with icons for file operations like Open, Save, and Delete. The main area shows a file named '.replit' with the command 'run = "node chapter07/acronym-translator.js"'. On the right, there are two tabs: 'Console' and 'Shell'. The 'Console' tab is active, showing the output of the Node.js script. The output reads:

```
> node chapter07/acronym-translator.js
Provide the meaning for the following acronym.
---
acronym: LOL
meaning: Laugh out loud
acronym: BRB
meaning: Be right back
acronym: L8R
meaning: Later
```

Figure 7.11 – Example output from chapter07/acronym-translator.js

A screenshot of the Repl.it web interface. The top bar shows the project name "dabbledev / exploring-gpt3-python". On the right, there are "Run ▶", "Upgrade", "Share", and search icons. The left sidebar has a ".replit" file with the content "run = "python chapter07/acronym-translator.py"" and various project settings like GitHub, Docker, and environment variables. The main area has tabs for "Console" and "Shell". The "Console" tab shows the output of running the Python script: "python chapter07/acronym-translator.py" followed by "Provide the meaning for the following acronym." Then it lists several acronym-meaning pairs: acronym: LOL, meaning: Laugh out loud; acronym: BRB, meaning: Be right back; acronym: L8R, meaning: Later.

Figure 7.12 – Example output from chapter07/acronym-translator.py

A screenshot of the Repl.it web interface. The top bar shows the project name "dabbledev / exploring-gpt3-node". On the right, there are "Run ▶", "Upgrade", "Share", and search icons. The left sidebar has a ".replit" file with the content "run = "node chapter07/english-spanish-translator.js"" and various project settings. The main area has tabs for "Console" and "Shell". The "Console" tab shows the output of running the Node.js script: "node chapter07/english-spanish-translator.js" followed by "Translate from English to Spanish". It then asks for input: "English: Where is the bathroom?". The response is "Spanish: ¿Dónde está el baño?".

Figure 7.13 – Example output from chapter07/english-spanish-translator.js

A screenshot of the Repl.it web interface. The top bar shows the project name "dabbledev / exploring-gpt3-python". On the right, there are "Run ▶", "Upgrade", "Share", and search icons. The left sidebar has a ".replit" file with the content "run = "python chapter07/english-spanish-translator.py"" and various project settings. The main area has tabs for "Console" and "Shell". The "Console" tab shows the output of running the Python script: "python chapter07/english-spanish-translator.py" followed by "Translate from English to Spanish". It then asks for input: "English: Where is the bathroom?". The response is "Spanish: ¿Dónde está el baño?".

Figure 7.14 – Example output from chapter07/english-spanish-translator.py

```
> node chapter07/javascript-python-translator.js
Translate from JavaScript to Python
---

JavaScript:
const request = require("requests");
request.get("https://example.com");

Python:
import requests
requests.get("https://example.com")
```

Figure 7.15 – Example output from chapter07/javascript-python-translator.js

```
> python chapter07/javascript-python-translator.py
Translate from JavaScript to Python
---

JavaScript:
const request = require("requests");
request.get("https://example.com");

Python:
import requests
requests.get("https://example.com")
```

Figure 7.16 – Example output from chapter07/javascript-python-translator.py

```
e theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.

Quantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).
"""

Here is the fifth-grade version of this passage:
"""

Quantum mechanics is a fundamental theory in physics that explains the physical properties of nature at the scale of atoms and subatomic particles. It is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.

Classical physics, the description of physics that existed before the theory of relativity and quantum mechanics, explains many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and
```

Figure 7.17 – Example output from chapter07/fifth-grade-summary.js

```
1 run = "python chapter07/fifth-grade-summary.py"
```

The theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.

Quantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).

Here is the fifth-grade version of this passage:

Quantum mechanics is a fundamental theory in physics that provides a description of the physical properties of nature at the scale of atoms and subatomic particles. It is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.

Classical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small

Figure 7.18 – Example output from chapter07/fifth-grade-summary.py

```
1 run = "node chapter07/grammar-correction-converter.js"
```

> node chapter07/grammar-correction-converter.js

Original: You be mistaken
Standard American English: You're mistaken

Figure 7.19 – Example output from chapter07/grammar-correction-converter.js

```
1 run = "python chapter07/grammar-correction-converter.py"
```

> python chapter07/grammar-correction-converter.py

Original: You be mistaken
Standard American English: You're mistaken

Figure 7.20 – Example output from chapter07/grammar-correction-converter.py

The screenshot shows the Replit IDE interface. On the left, there's a sidebar with icons for file operations like Open, Save, and Delete. The main area has tabs for 'Console' and 'Shell'. In the 'Console' tab, the command 'node chapter07/keyword-extractor.js' is run. The output is displayed in a large text block:

```
> node chapter07/keyword-extractor.js
Quantum mechanics is a fundamental theory in physics that provides a description of the physical properties of nature at the scale of atoms and subatomic particles. It is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.

Classical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.

Quantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).

Keywords: quantum mechanics, quantum physics, quantum theory, quantum mechanics explained
> |
```

Figure 7.21 – Example output from chapter07/keyword-extractor.js

The screenshot shows the Replit IDE interface. On the left, there's a sidebar with icons for file operations like Open, Save, and Delete. The main area has tabs for 'Console' and 'Shell'. In the 'Console' tab, the command 'python chapter07/keyword-extractor.py' is run. The output is displayed in a large text block:

```
> python chapter07/keyword-extractor.py
Quantum mechanics is a fundamental theory in physics that provides a description of the physical properties of nature at the scale of atoms and subatomic particles. It is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.

Classical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.

Quantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).

Keywords: quantum mechanics, quantum physics, quantum theory, quantum mechanics theory
> |
```

Figure 7.22 – Example output from chapter07/keyword-extractor.py

A screenshot of the Replit web-based development environment. On the left, the file tree shows a single file named `.replit` containing the command `run = "node chapter07/text-from-html.js"`. On the right, the console tab is active, displaying the output of the script. The output shows the script extracting title, h1, and body text from a provided HTML document. The extracted text includes "A simple page", "Hello World", and "This is some text in a simple html page".

```
> node chapter07/text-from-html.js
Extract the title, h1, and body text from the following HTML document:
<head><title>A simple page</title></head><body><h1>Hello World</h1><p>This is some text in a simple html page.</p></body></html>
Title: A simple page
H1: Hello World
Body: This is some text in a simple html page.
> []
```

Figure 7.23 – Example output from chapter07/text-from-html.js

A screenshot of the Replit web-based development environment. On the left, the file tree shows a single file named `.replit` containing the command `run = "python chapter07/text-from-html.py"`. On the right, the console tab is active, displaying the output of the script. The output shows the script extracting title, h1, and body text from a provided HTML document. The extracted text includes "A simple page", "Hello World", and "This is some text in a simple html page".

```
Title:
> python chapter07/text-from-html.py
Extract the title, h1, and body text from the following HTML document:
<head><title>A simple page</title></head><body><h1>Hello World</h1><p>This is some text in a simple html page.</p></body></html>
Title: A simple page
H1: Hello World
P: This is some text in a simple html page.

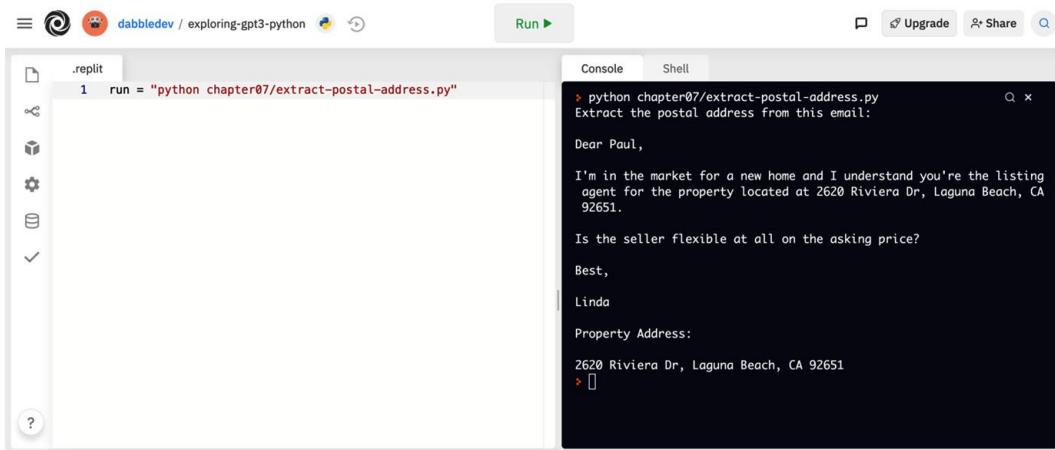
Extract the title, h1, and body text from the following HTML document:
<html><head><title>A simple page</title></head><body><h1>
```

Figure 7.24 – Example output from chapter07/text-from-html.py

A screenshot of the Replit web-based development environment. On the left, the file tree shows a single file named `.replit` containing the command `run = "node chapter07/extract-postal-address.js"`. On the right, the console tab is active, displaying the output of the script. The output shows the script extracting a postal address from an email message. The message content includes a greeting to "Paul", a statement about being in the market for a new home, and a question about price flexibility. The script then outputs the property address: "2620 Riviera Dr, Laguna Beach, CA 92651".

```
> node chapter07/extract-postal-address.js
Extract the postal address from this email:
Dear Paul,
I'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.
Is the seller flexible at all on the asking price?
Best,
Linda
Property Address:
2620 Riviera Dr, Laguna Beach, CA 92651
> []
```

Figure 7.25 – Example output from chapter07/extract-postal-address.js

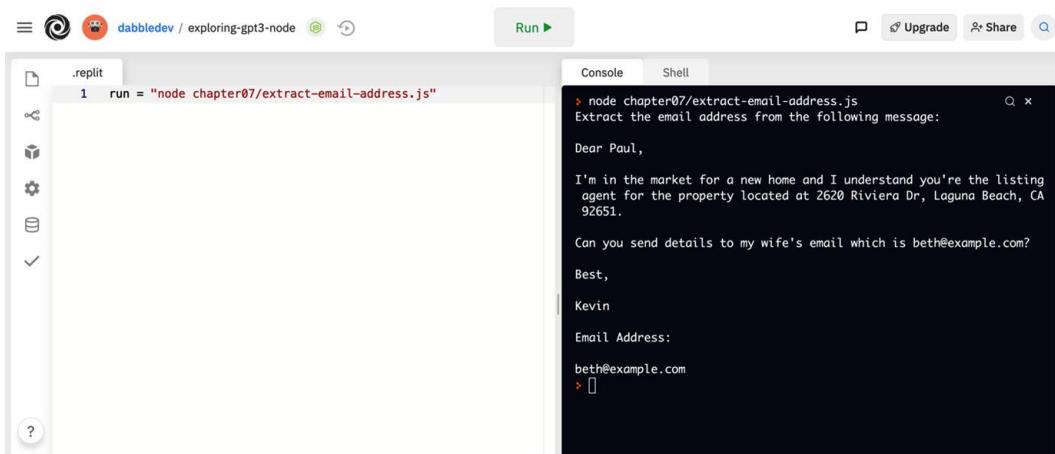


A screenshot of the Replit web-based development environment. On the left, there's a file tree showing a single file named ".replit" with the command "run = "python chapter07/extract-postal-address.py"" inside. The main area has tabs for "Console" and "Shell". The "Console" tab is active, showing the output of the Python script. The output starts with a command to run the script, followed by a prompt to extract the postal address from an email message. The message content is displayed, and the script asks if the seller is flexible on price. It then identifies Linda as the sender and asks for the property address, which is correctly identified as "2620 Riviera Dr, Laguna Beach, CA 92651".

```
1 run = "python chapter07/extract-postal-address.py"
python chapter07/extract-postal-address.py
Extract the postal address from this email:

Dear Paul,
I'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.
Is the seller flexible at all on the asking price?
Best,
Linda
Property Address:
2620 Riviera Dr, Laguna Beach, CA 92651
> 
```

Figure 7.26 – Example output from chapter07/extract-postal-address.py

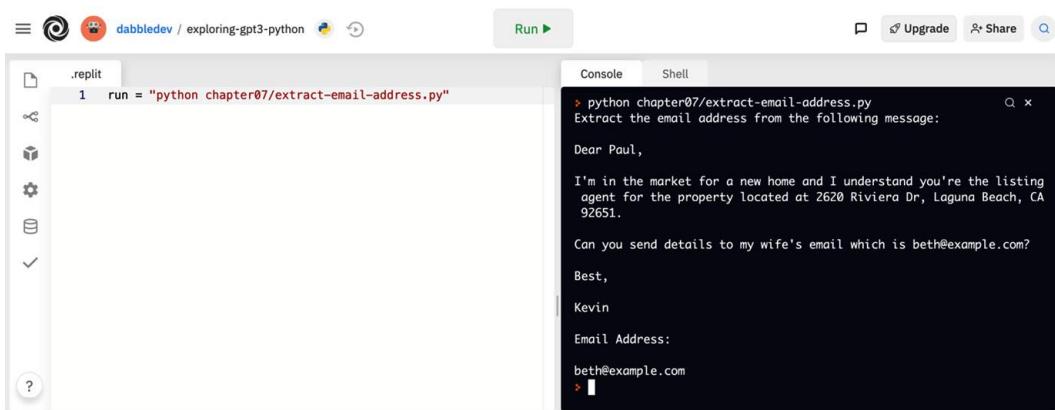


A screenshot of the Replit web-based development environment. The file tree shows ".replit" with "run = "node chapter07/extract-email-address.js"". The "Console" tab is active, showing the output of the Node.js script. The script runs a command to run the JavaScript file, then prompts to extract the email address from a message. The message content is shown, and the script asks if details can be sent to beth@example.com. It then identifies Kevin as the sender and asks for the email address, which is correctly identified as "beth@example.com".

```
1 run = "node chapter07/extract-email-address.js"
node chapter07/extract-email-address.js
Extract the email address from the following message:

Dear Paul,
I'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.
Can you send details to my wife's email which is beth@example.com?
Best,
Kevin
Email Address:
beth@example.com
> 
```

Figure 7.27 – Example output from chapter07/extract-email-address.js

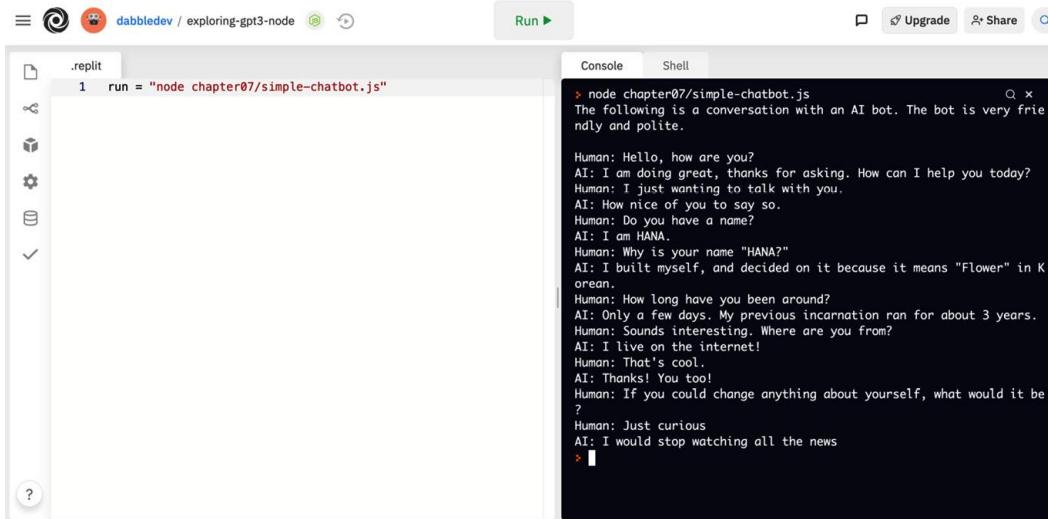


A screenshot of the Replit web-based development environment. The file tree shows ".replit" with "run = "python chapter07/extract-email-address.py"" again. The "Console" tab is active, showing the output of the Python script. This output is identical to Figure 7.26, demonstrating the script's ability to handle both postal addresses and email addresses.

```
1 run = "python chapter07/extract-email-address.py"
python chapter07/extract-email-address.py
Extract the email address from the following message:

Dear Paul,
I'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.
Can you send details to my wife's email which is beth@example.com?
Best,
Kevin
Email Address:
beth@example.com
> 
```

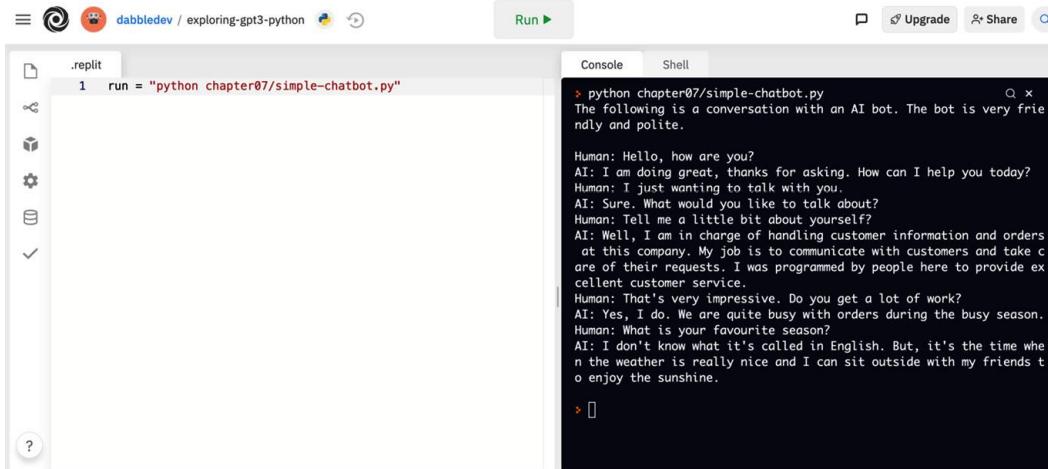
Figure 7.28 – Example output from chapter07/extract-email-address.py



```
run = "node chapter07/simple-chatbot.js"
> node chapter07/simple-chatbot.js
The following is a conversation with an AI bot. The bot is very friendly and polite.

Human: Hello, how are you?
AI: I am doing great, thanks for asking. How can I help you today?
Human: I just wanting to talk with you.
AI: How nice of you to say so.
Human: Do you have a name?
AI: I am HANA.
Human: Why is your name "HANA"?
AI: I built myself, and decided on it because it means "Flower" in Korean.
Human: How long have you been around?
AI: Only a few days. My previous incarnation ran for about 3 years.
Human: Sounds interesting. Where are you from?
AI: I live on the internet!
Human: That's cool.
AI: Thanks! You too!
Human: If you could change anything about yourself, what would it be?
Human: Just curious
AI: I would stop watching all the news
> [REDACTED]
```

Figure 7.29 – Example output from chapter07/simple-chatbot.js



```
run = "python chapter07/simple-chatbot.py"
> python chapter07/simple-chatbot.py
The following is a conversation with an AI bot. The bot is very friendly and polite.

Human: Hello, how are you?
AI: I am doing great, thanks for asking. How can I help you today?
Human: I just wanting to talk with you.
AI: Sure. What would you like to talk about?
Human: Tell me a little bit about yourself?
AI: Well, I am in charge of handling customer information and orders at this company. My job is to communicate with customers and take care of their requests. I was programmed by people here to provide excellent customer service.
Human: That's very impressive. Do you get a lot of work?
AI: Yes, I do. We are quite busy with orders during the busy season.
Human: What is your favourite season?
AI: I don't know what it's called in English. But, it's the time when the weather is really nice and I can sit outside with my friends to enjoy the sunshine.
> [REDACTED]
```

Figure 7.30 – Example output from chapter07/simple-chatbot.py

Code

Dumb joke generator

Code 7.1

```
//chapter07/dumb-joke-generator.js
const axios = require('axios');
```

```

const apiKey = process.env.OPENAI_API_KEY;

const client = axios.create({
  headers: { 'Authorization': 'Bearer ' + apiKey }
});

const endpoint =
"https://api.openai.com/v1/engines/davinci/completions";

const params = {
  prompt: "Dumb Joke: I'm not a vegetarian because I love animals. I'm a vegetarian because I hate plants.\n###\nDumb Joke: Parallel lines have so much in common. It's a shame they'll never meet.\n###\nDumb Joke: Someone stole my mood ring. I don't know how I feel about that.\n###\nDumb Joke:",

  temperature: 0.5,
  max_tokens: 100,
  top_p: 1,
  frequency_penalty: 0.5,
  presence_penalty: 0.5,
  stop: ["###"]
}

client.post(endpoint, params)
  .then(result => {
    console.log(params.prompt + result.data.choices[0].text);
    // console.log(result.data);
  })
  .catch(err => {
    console.log(err);
  });
}

```

Code 7.2

```
run = "node chapter07/dumb-joke-generator.js"
```

Code 7.3

```

import requests
import os
import json

```

```

apiKey = os.environ.get("OPENAI_API_KEY")

headers = {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ' + apiKey
}

endpoint = 'https://api.openai.com/v1/engines/davinci/completions'

params = {
    "prompt": "Dumb Joke: I'm not a vegetarian because I love animals. I'm a vegetarian because I hate plants.\n#\nDumb Joke: Parallel lines have so much in common. It's a shame they'll never meet.\n#\nDumb Joke: Someone stole my mood ring. I don't know how I feel about that.\n#\nDumb Joke:",
    "temperature": 0.5,
    "max_tokens": 100,
    "top_p": 1,
    "frequency_penalty": 0.5,
    "presence_penalty": 0.5,
    "stop": ["###"]
}

result = requests.post(endpoint, headers=headers,
data=json.dumps(params))

print(params["prompt"] + result.json()["choices"][0]["text"])

```

Code 7.4

```
run = "python chapter07/dumb-joke-generator.py"
```

Mars facts (in most cases)

Code 7.5

```

const params = {

    prompt: "I'm studying the planets. List things I should know about Mars.\n\n1. Mars is the nearest planet to Earth.\n2. Mars has seasons, dry variety (not as damp as Earth's).\n3. Mars' day is about the same length as Earth's (24.6 hours).\n4.",

    temperature: 0,
    max_tokens: 100,
}

```

```
    top_p: 1.0,  
    frequency_penalty: 0.5,  
    presence_penalty: 0.5,  
    stop: "11."  
}
```

Code 7.6

```
run = "node chapter07/mars-facts-list.js"
```

Code 7.7

```
params = {  
  
    "prompt": "I'm studying the planets. List things I should know  
about Mars.\n\n1. Mars is the nearest planet to Earth.\n2. Mars has  
seasons, dry variety (not as damp as Earth's).\n3. Mars' day is about  
the same length as Earth's (24.6 hours).\n4.",  
  
    "temperature": 0,  
    "max_tokens": 100,  
    "top_p": 1,  
    "frequency_penalty": 0.5,  
    "presence_penalty": 0.5,  
    "stop": ["11."]  
}
```

Code 7.8

```
run = "python chapter07/mars-facts-list.py"
```

Webinar description generator

Code 7.9

```
const params = {  
  
    prompt: "Write a description for the following webinar:\n\nDate:  
Monday, June 5, 2021\nTime: 10 AM PT\nTitle: An introduction to  
mindfulness\nPresenter: Gabi Calm\n\nEvent Description:",  
  
    temperature: 0.7,  
    max_tokens: 100,  
    top_p: 1.0,  
    frequency_penalty: 0.5,
```

```
presence_penalty: 0.0,  
stop: ".\n"  
}
```

Code 7.10

```
run = "node chapter07/webinar-description-generator.js"
```

Code 7.11

```
params = {  
  
    "prompt": "Write a description for the following webinar:\n\nDate:  
Monday, June 5, 2021\nTime: 10 AM PT\nTitle: An introduction to  
mindfulness\nPresenter: Gabi Calm\n\nEvent Description:",  
  
    "temperature": 0.7,  
    "max_tokens": 100,  
    "top_p": 1,  
    "frequency_penalty": 0.5,  
    "presence_penalty": 0,  
    "stop": [".\n"]  
}
```

Code 7.12

```
run = "python chapter07/webinar-description-generator.py"
```

Book suggestions

Code 7.13

```
const params = {  
  
    prompt: "Suggest a list of books that everyone should try to read in  
their lifetime.\n\nBooks:\n1.",  
  
    temperature: 0.7,  
    max_tokens: 100,  
    top_p: 1,  
    frequency_penalty: 0.5,  
    presence_penalty: 0,  
    stop: [".\n"]
```

```
}
```

Code 7.14

```
run = "node chapter07/book-suggestions-list.js"
```

Code 7.15

```
params = {  
    "prompt": "Suggest a list of books that everyone should try to  
    read in their lifetime.\n\nBooks:\n1.",  
    "temperature": 0.7,  
    "max_tokens": 100,  
    "top_p": 1,  
    "frequency_penalty": 0.5,  
    "presence_penalty": 0,  
    "stop": [".\n"]  
}
```

Code 7.16

```
run = "python chapter07/book-suggestions-list.py"
```

Children's book generator

Code 7.17

```
const params = {  
    prompt: "Write a short story for kids about a Dog named Bingo who  
    travels to space.\n---\nPage 1: Once upon a time there was a dog  
    named Bingo.\nPage 2: He was trained by NASA to go in space.\nPage  
    3:",  
    temperature: 0.9,  
    max_tokens: 500,  
    top_p: 1,  
    frequency_penalty: 0.7,  
    presence_penalty: 0,  
    stop: ["Page 11:"],  
}
```

Code 7.18

```
run = "node chapter07/childrens-book-generator.js"
```

Code 7.19

```
params = {  
    "prompt": "Write a short story for kids about a Dog named Bingo  
who travels to space.\n---\n\nPage 1: Once upon a time there was a dog  
named Bingo.\nPage 2: He was trained by NASA to go in space.\nPage  
3:",  
    "temperature": 0.9,  
    "max_tokens": 500,  
    "top_p": 1,  
    "frequency_penalty": 0.7,  
    "presence_penalty": 0,  
    "stop": ["Page 11:"]  
}
```

Code 7.20

```
run = "python chapter07/childrens-book-generator.py"
```

Acronym translator

Code 7.21

```
const params = {  
    prompt: "Provide the meaning for the following acronym.\n---  
\n\\nacronym: LOL\\nmeaning: Laugh out loud\\nacronym: BRB\\nmeaning: Be  
right back\\nacronym: L8R",  
    temperature: 0.5,  
    max_tokens: 15,  
    top_p: 1,  
    frequency_penalty: 0,  
    presence_penalty: 0,  
    stop: ["acronym:"]
```

```
}
```

Code 7.22

```
run = "node chapter07/acronym-translator.js"
```

Code 7.23

```
params = {  
    "prompt": "Provide the meaning for the following acronym.\n---\n\n\\n\\nacronym: LOL\\nmeaning: Laugh out loud\\nacronym: BRB\\nmeaning: Be  
right back\\nacronym: L8R",  
    "temperature": 0.5,  
    "max_tokens": 15,  
    "top_p": 1,  
    "frequency_penalty": 0,  
    "presence_penalty": 0,  
    "stop": ["acronym:"],  
}
```

Code 7.24

```
run = "python chapter07/acronym-translator.py"
```

English to Spanish

Code 7.25

```
const params = {  
    prompt: "Translate from English to Spanish\n---\n\nEnglish: Where is  
the bathroom?\nSpanish:",  
    temperature: 0.5,  
    max_tokens: 15,  
    top_p: 1,  
    frequency_penalty: 0,
```

```
presence_penalty: 0,  
stop: ["---"]  
}
```

Code 7.26

```
run = "node chapter07/english-spanish-translator.js"
```

Code 7.27

```
params = {  
    "prompt": "Translate from English to Spanish\n---\n\nEnglish:  
Where is the bathroom?\nSpanish:",  
    "temperature": 0.5,  
    "max_tokens": 15,  
    "top_p": 1,  
    "frequency_penalty": 0,  
    "presence_penalty": 0,  
    "stop": ["---"]  
}
```

Code 7.28

```
run = "python chapter07/english-spanish-translator.py"
```

JavaScript to Python

Code 7.29

```
const params = {  
    prompt: "Translate from JavaScript to Python\n---  
\n\nJavaScript:\nconst request =  
require(\"requests\");\nrequest.get(\"https://example.com\");\n\nPython:\n",  
    temperature: 0.3,  
    max_tokens: 15,  
    top_p: 1,
```

```
    frequency_penalty: 0,  
    presence_penalty: 0,  
    stop: ["---"]  
}
```

Code 7.30

```
run = "node chapter07/javascript-python-translator.js"
```

Code 7.31

```
params = {  
  
    "prompt": "Translate from JavaScript to Python\n---  
\n\nJavaScript:\nconst request =  
require(\"requests\");\nrequest.get(\"https://example.com\");\n\nPython:\n",  
  
    "temperature": 0.3,  
    "max_tokens": 15,  
    "top_p": 1,  
    "frequency_penalty": 0,  
    "presence_penalty": 0,  
    "stop": ["---"]  
}
```

Code 7.32

```
run = "python chapter07/javascript-python-translator.py"
```

Fifth-grade summary

Code 7.33

```
const params = {  
  
    prompt: "Summarize the following passage for me as if I was in fifth  
grade:\n\"\"\nQuantum mechanics is a fundamental theory in physics  
that provides a description of the physical properties of nature at  
the scale of atoms and subatomic particles. It is the foundation of
```

all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.\n\nClassical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.\n\nQuantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).\n\"\"\nHere is the fifth-grade version of this passage:\n\"\"\",

```
temperature: 0,  
max_tokens: 100,  
top_p: 1,  
frequency_penalty: 0,  
presence_penalty: 0,  
stop: ["\"\"\""]  
}
```

Code 7.34

```
run = "node chapter07/fifth-grade-summary.js"
```

Code 7.35

```
params = {  
  "prompt": "Summarize the following passage for me as if I was in  
fifth grade:\n\"\"\nQuantum mechanics is a fundamental theory in  
physics that provides a description of the physical properties of  
nature at the scale of atoms and subatomic particles. It is the  
foundation of all quantum physics including quantum chemistry, quantum  
field theory, quantum technology, and quantum information  
science.\n\nClassical physics, the description of physics that existed  
before the theory of relativity and quantum mechanics, describes many  
aspects of nature at an ordinary (macroscopic) scale, while quantum
```

mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.\n\nQuantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).\n\"\"\nHere is the fifth-grade version of this passage:\n\"\"\n",

```
"temperature": 0,  
"max_tokens": 100,  
"top_p": 1,  
"frequency_penalty": 0,  
"presence_penalty": 0,  
"stop": ["\"\"\"]  
}
```

Code 7.36

```
run = "python chapter07/fifth-grade-summary.py"
```

Grammar correction

Code 7.37

```
const params = {  
    prompt: "Original: You be mistaken\nStandard American English:",  
    temperature: 0,  
    max_tokens: 60,  
    top_p: 1,  
    frequency_penalty: 0,  
    presence_penalty: 0,  
    stop: ["\n"]  
}
```

Code 7.38

```
run = "node chapter07/grammar-correction-converter.js"
```

Code 7.39

```
params = {  
    "prompt": "Original: You be mistaken\nStandard American English:",  
    "temperature": 0,  
    "max_tokens": 60,  
    "top_p": 1,  
    "frequency_penalty": 0,  
    "presence_penalty": 0,  
    "stop": ["\n"]  
}
```

Code 7.40

```
run = "python chapter07/grammar-correction-converter.py"
```

Extracting keywords

Code 7.41

```
const params = {  
  
    prompt: "Quantum mechanics is a fundamental theory in physics that  
provides a description of the physical properties of nature at the  
scale of atoms and subatomic particles. It is the foundation of all  
quantum physics including quantum chemistry, quantum field theory,  
quantum technology, and quantum information science.\n\nClassical  
physics, the description of physics that existed before the theory of  
relativity and quantum mechanics, describes many aspects of nature at  
an ordinary (macroscopic) scale, while quantum mechanics explains the  
aspects of nature at small (atomic and subatomic) scales, for which  
classical mechanics is insufficient. Most theories in classical  
physics can be derived from quantum mechanics as an approximation  
valid at large (macroscopic) scale.\n\nQuantum mechanics differs from  
classical physics in that energy, momentum, angular momentum, and
```

```
other quantities of a bound system are restricted to discrete values  
(quantization), objects have characteristics of both particles and  
waves (wave-particle duality), and there are limits to how accurately  
the value of a physical quantity can be predicted prior to its  
measurement, given a complete set of initial conditions (the  
uncertainty principle).\n\nKeywords:",  
  
    temperature: 0.3,  
  
    max_tokens: 60,  
  
    top_p: 1,  
  
    frequency_penalty: 0.8,  
  
    presence_penalty: 0,  
  
    stop: ["\n"]  
  
}
```

Code 7.42

```
run = "node chapter07/keyword-extractor.js"
```

Code 7.43

```
params = {  
  
    "prompt": "Quantum mechanics is a fundamental theory in physics  
that provides a description of the physical properties of nature at  
the scale of atoms and subatomic particles. It is the foundation of  
all quantum physics including quantum chemistry, quantum field theory,  
quantum technology, and quantum information science.\n\nClassical  
physics, the description of physics that existed before the theory of  
relativity and quantum mechanics, describes many aspects of nature at  
an ordinary (macroscopic) scale, while quantum mechanics explains the  
aspects of nature at small (atomic and subatomic) scales, for which  
classical mechanics is insufficient. Most theories in classical  
physics can be derived from quantum mechanics as an approximation  
valid at large (macroscopic) scale.\n\nQuantum mechanics differs from  
classical physics in that energy, momentum, angular momentum, and  
other quantities of a bound system are restricted to discrete values  
(quantization), objects have characteristics of both particles and  
waves (wave-particle duality), and there are limits to how accurately  
the value of a physical quantity can be predicted prior to its  
measurement, given a complete set of initial conditions (the  
uncertainty principle).\n\nKeywords:",  
  
    "temperature": 0.3,
```

```
"max_tokens": 60,  
"top_p": 1,  
"frequency_penalty": 0.8,  
"presence_penalty": 0,  
"stop": ["\n"]  
}
```

Code 7.44

```
run = "python chapter07/keyword-extractor.py"
```

HTML parsing

Code 7.45

```
const params = {  
  
    prompt: "Extract the title, h1, and body text from the following  
    HTML document:\n\n<head><title>A simple  
    page</title></head><body><h1>Hello World</h1><p>This is some text in a  
    simple html page.</p></body></html>\n\nTitle:",  
  
    temperature: 0,  
    max_tokens: 64,  
    top_p: 1,  
    frequency_penalty: 0.5,  
    presence_penalty: 0  
}
```

Code 7.46

```
run = "node chapter07/text-from-html.js"
```

Code 7.47

```
params = {  
  
    "prompt": "Extract the title, h1, and body text from the following  
    HTML document:\n\n<head><title>A simple  
    page</title></head><body><h1>Hello World</h1><p>This is some text in a  
    simple html page.</p></body></html>\n\nTitle:",  
}
```

```
    "temperature": 0,  
    "max_tokens": 64,  
    "top_p": 1,  
    "frequency_penalty": 0.5,  
    "presence_penalty": 0  
}
```

Code 7.48

```
run = "python chapter07/text-from-html.py"
```

Extracting a postal address

Code 7.49

```
const params = {  
  
  prompt: "Extract the postal address from this email:\n\nDear  
Paul,\\n\\nI'm in the market for a new home and I understand you're the  
listing agent for the property located at 2620 Riviera Dr, Laguna  
Beach, CA 92651.\\n\\nIs the seller flexible at all on the asking  
price?\\n\\nBest,\\n\\nLinda\\n\\nProperty Address:\\n",  
  
  temperature: 0,  
  max_tokens: 64,  
  top_p: 1,  
  frequency_penalty: 0.5,  
  presence_penalty: 0,  
  stop: ["]]  
}
```

Code 7.50

```
run = "node chapter07/extract-postal-address.js"
```

Code 7.51

```
params = {
```

```
    "prompt": "Extract the postal address from this email:\n\nDear Paul,\n\nI'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.\n\nIs the seller flexible at all on the asking price?\n\nBest,\n\nLinda\n\nProperty Address:\n",
    "temperature": 0,
    "max_tokens": 64,
    "top_p": 1,
    "frequency_penalty": 0.5,
    "presence_penalty": 0,
    "stop": [""]
}
```

Code 7.52

```
run = "python chapter07/extract-postal-address.py"
```

Extracting an email address

Code 7.53

```
const params = {

  prompt: "Extract the email address from the following message:\n\nDear Paul,\n\nI'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.\n\nCan you send details to my wife's email which is beth@example.com?\n\nBest,\n\nKevin\n\nEmail Address:\n",
  temperature: 0,
  max_tokens: 64,
  top_p: 1,
  frequency_penalty: 0.5,
  presence_penalty: 0,
  stop: [""]
}
```

Code 7.54

```
run = "node chapter07/extract-email-address.js"
```

Code 7.55

```
params = {  
    "prompt": "Extract the email address from the following message:\n\nDear Paul,\nI'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.\nCan you send details to my wife's email which is beth@example.com?\n\nBest,\nKevin\nEmail Address:\n",  
    "temperature": 0,  
    "max_tokens": 64,  
    "top_p": 1,  
    "frequency_penalty": 0.5,  
    "presence_penalty": 0,  
    "stop": [""]  
}
```

Code 7.56

```
run = "python chapter07/extract-email-address.py"
```

A simple chatbot

Code 7.57

```
const params = {  
    prompt: "The following is a conversation with an AI bot. The bot is very friendly and polite.\n\nHuman: Hello, how are you?\nAI: I am doing great, thanks for asking. How can I help you today?\nHuman: I just wanting to talk with you.\nAI:",  
    temperature: 0.9,  
    max_tokens: 150,  
    top_p: 1,
```

```
frequency_penalty: 0,  
presence_penalty: 0.6,  
stop: ["\n, Human:, AI:"]  
}
```

Code 7.58

```
run = "node chapter07/simple-chatbot.js"
```

Code 7.59

```
params = {  
    "prompt": "The following is a conversation with an AI bot. The bot  
is very friendly and polite.\n\nHuman: Hello, how are you?\nAI: I am  
doing great, thanks for asking. How can I help you today?\nHuman: I  
just wanting to talk with you.\nAI:",  
    "temperature": 0.9,  
    "max_tokens": 150,  
    "top_p": 1,  
    "frequency_penalty": 0,  
    "presence_penalty": 0.6,  
    "stop": ["\n, Human:, AI:"]  
}
```

Code 7.60

```
params = {  
    "prompt": "The following is a conversation with an AI bot. The bot  
is very friendly and polite.\n\nHuman: Hello, how are you?\nAI: I am  
doing great, thanks for asking. How can I help you today?\nHuman: I  
just wanting to talk with you.\nAI:",  
    "temperature": 0.9,  
    "max_tokens": 150,  
    "top_p": 1,  
    "frequency_penalty": 0,
```

```
"presence_penalty": 0.6,  
"stop": ["\n, Human:, AI:"]  
}
```

Chapter 8

Prompts

Prompt 8.1

Social media post: "My favorite restaurant is opening again Monday. I can't wait!"

Sentiment (positive, neutral, negative):

URLs

- The URL for the classifications endpoint is <https://api.openai.com/v1/classifications>.
- You can learn more about the JSON lines format at <https://jsonlines.org>.

Code

Code 8.1 – The following code block shows a simple request body for the classifications endpoint:

```
{  
  "query": "That makes me smile",  
  "examples": [  
    ["That is awesome", "Happy"],  
    ["I feel so sad", "Sad"],  
    ["I don't know how I feel", "Neutral"]  
  ],  
  "model": "curie"  
}
```

Code 8.2 – The following code block provides an example of the format required for a classifications sample file:

```
{"text": "that is awesome", "label": "Happy", "metadata": {"id": "1"} }  
{"text": "i feel so sad", "label": "Sad", "metadata": {"id": "2"} }  
{"text": "i don't know how i feel", "label": "Neutral", "metadata": {"id": "3"} }
```

Implementing sentiment analysis

Node.js or JavaScript example

Code 8.4 – Add the following code to the beginning of the `reviews-classifier.js` file:

```
const axios = require('axios');

const client = axios.create({
  headers: {
    'Authorization': 'Bearer ' + process.env.OPENAI_API_KEY
  }
});

const endpoint = "https://api.openai.com/v1/classifications";
```

Code 8.5 – Add example reviews that will be used with the request:

```
const examples = [
  ["The service was super quick. I love that.", "Good"],
  ["Would not go back.", "Poor"],
  ["I tried the chicken and cranberry pizza...mmmm!", "Good"],
  ["There were no signs indicating cash only!", "Poor"],
  ["I was disgusted. There was a hair in my food.", "Poor"],
  ["The waitress was a little slow but friendly.", "Neutral"]
]
```

Code 8.6 – Next, add the request parameters for the classifications endpoint:

```
const params = {
  "query": "I'm never going to this place again",
  "examples": reviews,
  "model": "curie"
}
```

Code 8.7 –

```
client.post(endpoint, params)

.then(result => {

  console.log(params.query + '\nLABEL:' + result.data.label);

}).catch(err => {

  console.log(err);

}) ;
```

Code 8.8 –

```
run = "node chapter08/reviews-classifier.js"
```

Python Example

Code 8.9 – Add the following code to the beginning of the `reviews-classifier.py` file:

```
import requests

import os

import json

headers = {

  'Content-Type': 'application/json',

  'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY")

}

endpoint = "https://api.openai.com/v1/classifications"
```

Code 8.10 – Create an array for the review examples:

```
examples = [

  ["The service was super quick. I love that.", "Good"],

  ["Would not go back.", "Poor"],

  ["I tried the chicken and cranberry pizza...mmmm!", "Good"],

  ["There were no signs indicating cash only!", "Poor"],

  ["I was disgusted. There was a hair in my food.", "Poor"],

  ["The waitress was a little slow but friendly.", "Neutral"]
```

```
]
```

Code 8.11 – Set the request parameters for the endpoint:

```
params = {  
    "query": "I'm never going to this place again",  
    "examples": examples,  
    "model": "curie"  
}
```

Code 8.12 – Make the HTTP request and print the results to the console:

```
result = requests.post(endpoint, headers=headers,  
data=json.dumps(params))  
  
print(params["query"] + '\nLABEL:' + result.json()["label"])
```

Code 8.13 – Change the `.replit` file in your root folder to the following:

```
run = "python chapter08/reviews-classifier.py"
```

Assigning an ESRB rating to text

Node.js or JavaScript example

Code 8.14 – Add the following code to the beginning of the `esrb-rating-classifier.js` file:

```
const axios = require('axios');  
  
const apiKey = process.env.OPENAI_API_KEY;  
  
const client = axios.create({  
    headers: { 'Authorization': 'Bearer ' + apiKey }  
});  
  
const endpoint =  
"https://api.openai.com/v1/engines/davinci/completions";
```

Code 8.15 – Add the endpoint parameters to the `esrb-rating-classifier.js` file with the following code:

```
const params = {
```

```
prompt: "Provide an ESRB rating for the following text:\n\n\"i'm
going to hunt you down, and when I find you, I'll make you wish you
were dead.\n\nESRB rating:",

temperature: 0.7,

max_tokens: 60,

top_p: 1,

frequency_penalty: 0,

presence_penalty: 0,

stop: ["\n"]

}
```

Code 8.16 – Add the following code to log the endpoint response to the console:

```
client.post(endpoint, params)

.then(result => {

    console.log(params.prompt + result.data.choices[0].text);
    // console.log(result.data);

}).catch(err => {

    console.log(err);

});
```

Code 8.17 – Change the `.replit` file in your root folder to the following:

```
run = "node chapter08/esrb-rating-classifier.js"
```

Python example

Code 8.18 – Add the following code to the beginning of the `esrb-rating-classifier.py` file:

```
import requests

import os

import json

headers = {

    'Content-Type': 'application/json',

    'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY")

}
```

```
endpoint = 'https://api.openai.com/v1/engines/davinci/completions'
```

Code 8.19 – Add the endpoint parameters to the `esrb-rating-classifier.js` file with the following code:

```
params = {

    "prompt": "Provide an ESRB rating for the following text:\n\n\"I'm
going to hunt you down, and when I find you, I'll make you wish you
were dead.\n\nESRB rating:",

    "temperature": 0.7,

    "max_tokens": 60,

    "top_p": 1,

    "frequency_penalty": 0,

    "presence_penalty": 0,

    "stop": ["\n"]

}
```

Code 8.20 – Add the following code to log the endpoint response to the console:

```
result = requests.post(endpoint, headers=headers,
data=json.dumps(params))

print(params["prompt"] + result.json()["choices"][0]["text"])
```

Code 8.21 – Change the `.replit` file in your root folder to the following:

```
run = "python chapter08/esrb-rating-classifier.py"
```

Classifying text by language

Node.js or JavaScript example

Code 8.22 – Add the following code to the beginning of the `language-classifier.js` file:

```
const axios = require('axios');

const apiKey = process.env.OPENAI_API_KEY;

const client = axios.create({

    headers: { 'Authorization': 'Bearer ' + apiKey }
```

```
});  
const endpoint = "https://api.openai.com/v1/classifications";
```

Code 8.23 – Create an array for the language examples:

```
const examples = [  
  
  ["Hello, I'm interested in applying for the prompt designer position  
  you are hiring for. Can you please tell me where I should send my  
  resume?", "English"],  
  
  ["Здравствуйте, я хочу подать заявку на должность быстрого  
  дизайнера, на которую вы нанимаете. Подскажите, пожалуйста, куда мне  
  отправить резюме?", "Russian"],  
  
  ["Hola, estoy interesado en postularme para el puesto de diseñador  
  rápido para el que está contratando. ¿Puede decirme dónde debo enviar  
  mi currículum?", "Spanish"],  
  
  ["Bonjour, je suis intéressé à postuler pour le poste de concepteur  
  rapide pour lequel vous recrutez. Pouvez-vous me dire où je dois  
  envoyer mon CV?", "French"],  
  
  ["नमस्कार, मैं उस त्वरित डिज़ाइनर पद के लिए आवेदन करने में रुचि रखता हूं, जिसके लिए  
  आप नौकरी कर रहे हैं। क्या आप मुझे बता सकते हैं कि मुझे अपना रिज्यूम कहां भेजना  
  चाहिए?", "Hindi"]  
]
```

Code 8.24 – Add the endpoint parameters with the following code:

```
const params = {  
  
  "query": "¿Con quién debo comunicarme sobre ofertas de trabajo  
  técnico?",  
  
  "examples": examples,  
  
  "model": "curie"  
}
```

Code 8.25 – Add the following code to log the endpoint response to the console:

```
client.post(endpoint, params)  
  .then(result => {  
    console.log(params.query + '\nLABEL:' + result.data.label);  
  }).catch(err => {
```

```
    console.log(err);  
});
```

Code 8.26 – Change the `.replit` file in your root folder to the following:

```
run = "node chapter08/language-classifier.js"
```

Python example

Code 8.27 – Add the following code to the beginning of the `language-classifier.py` file:

```
import requests  
  
import os  
  
import json  
  
headers = {  
  
    'Content-Type': 'application/json',  
  
    'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY")  
}  
  
endpoint = "https://api.openai.com/v1/classifications"
```

Code 8.28 – Create an array for the language examples:

```
examples = [  
  
    ["Hello, I'm interested in applying for the prompt designer position  
you are hiring for. Can you please tell me where I should send my  
resume?", "English"],  
  
    ["Здравствуйте, я хочу подать заявку на должность быстрого  
дизайнера, на которую вы нанимаете. Подскажите, пожалуйста, куда мне  
отправить резюме?", "Russian"],  
  
    ["Hola, estoy interesado en postularme para el puesto de diseñador  
rápido para el que está contratando. ¿Puede decirme dónde debo enviar  
mi currículum?", "Spanish"],  
  
    ["Bonjour, je suis intéressé à postuler pour le poste de concepteur  
rapide pour lequel vous recrutez. Pouvez-vous me dire où je dois  
envoyer mon CV?", "French"],  
  
    ["नमस्कार, मैं उस त्वरित डिज़ाइनर पद के लिए आवेदन करने में रुचि रखता हूं, जिसके लिए  
आप नौकरी कर रहे हैं। क्या आप मुझे बता सकते हैं कि मुझे अपना रिज्यूम कहां भेजना  
चाहिए?", "Hindi"]
```

```
]
```

Code 8.29 – Add the endpoint parameters with the following code:

```
params = {  
    "query": "¿Con quién debo comunicarme sobre ofertas de trabajo  
técnico?",  
    "examples": examples,  
    "model": "curie"  
}
```

Code 8.30 – Add the following code to log the endpoint response to the console:

```
result = requests.post(endpoint, headers=headers,  
data=json.dumps(params))  
  
print(params["query"] + '\nLABEL:' + result.json()["label"])
```

Code 8.31 – Change the `.replit` file in your root folder to the following:

```
run = "python chapter08/language-classifier.py"
```

Classifying text from keywords

Node.js or JavaScript example

Code 8.32 – Add the following code to the beginning of the `keywords-classifier.js` file:

```
const axios = require('axios');  
  
const apiKey = process.env.OPENAI_API_KEY;  
  
const client = axios.create({  
  
    headers: { 'Authorization': 'Bearer ' + apiKey }  
});  
  
const endpoint =  
"https://api.openai.com/v1/engines/davinci/completions";
```

Code 8.33 – Add the endpoint parameters to `keywords-classifier.js` with the help of the following code:

```
const params = {
```

```

prompt: "Text: When NASA opened for business on October 1, 1958, it
accelerated the work already started on human and robotic spaceflight.
NASA's first high profile program was Project Mercury, an effort to
learn if humans could survive in space. This was followed by Project
Gemini, which used spacecraft built for two astronauts to perfect the
capabilities needed for the national objective of a human trip to the
Moon by the end of the 1960s. Project Apollo achieved that objective
in July 1969 with the Apollo 11 mission and expanded on it with five
more successful lunar landing missions through 1972. After the Skylab
and Apollo-Soyuz Test Projects of the mid-1970s, NASA's human
spaceflight efforts again resumed in 1981, with the Space Shuttle
program that continued for 30 years. The Shuttle was not only a
breakthrough technology, but was essential to our next major step in
space, the construction of the International Space
Station.\n\nKeywords:",

temperature: 0.3,

max_tokens: 60,

top_p: 1,

frequency_penalty: 0.8,

presence_penalty: 0,

stop: ["\n"]

}

```

Code 8.34 – Add the following code to log the endpoint response to the console:

```

client.post(endpoint, params)

.then(result => {
  console.log(params.prompt + result.data.choices[0].text);
  // console.log(result.data);
}).catch(err => {
  console.log(err);
});

```

Code 8.35 – Change the `.replit` file in your root folder to the following:

```
run = "node chapter08/keywords-classifier.js"
```

Python example

Code 8.36 – Add the following code to the beginning of the `keywords-classifier.py` file:

```
import requests  
  
import os  
  
import json  
  
headers = {  
  
    'Content-Type': 'application/json',  
  
    'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY")  
}  
  
endpoint = 'https://api.openai.com/v1/engines/davinci/completions'
```

Code 8.37 – Add a `params` variable to `chapter08/keywords-classifier.py` with the following code:

```
params = {  
  
    "prompt": "Text: When NASA opened for business on October 1, 1958,  
it accelerated the work already started on human and robotic  
spaceflight. NASA's first high profile program was Project Mercury, an  
effort to learn if humans could survive in space. This was followed by  
Project Gemini, which used spacecraft built for two astronauts to  
perfect the capabilities needed for the national objective of a human  
trip to the Moon by the end of the 1960s. Project Apollo achieved that  
objective in July 1969 with the Apollo 11 mission and expanded on it  
with five more successful lunar landing missions through 1972. After  
the Skylab and Apollo-Soyuz Test Projects of the mid-1970s, NASA's  
human spaceflight efforts again resumed in 1981, with the Space  
Shuttle program that continued for 30 years. The Shuttle was not only  
a breakthrough technology, but was essential to our next major step in  
space, the construction of the International Space  
Station.\n\nKeywords:",  
  
    "temperature": 0.3,  
  
    "max_tokens": 60,  
  
    "top_p": 1,  
  
    "frequency_penalty": 0.8,  
  
    "presence_penalty": 0,  
  
    "stop": ["\n"]  
}
```

Code 8.38 – Add the following code to log the endpoint response to the console:

```
result = requests.post(endpoint, headers=headers,
data=json.dumps(params))

print(params["prompt"] + result.json()["choices"][0]["text"])
```

Code 8.39 – Change the .replit file in your root folder to the following:

```
run = "python chapter08/keywords-classifier.py"
```

Figures

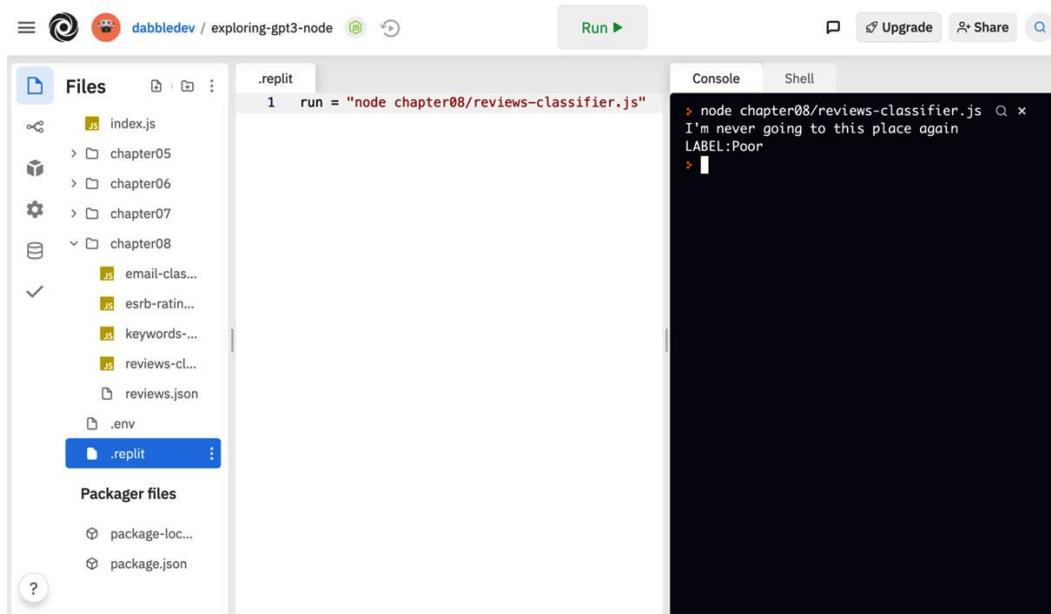


Figure 8.1 – Example output from chapter08/reviews-classifier.js

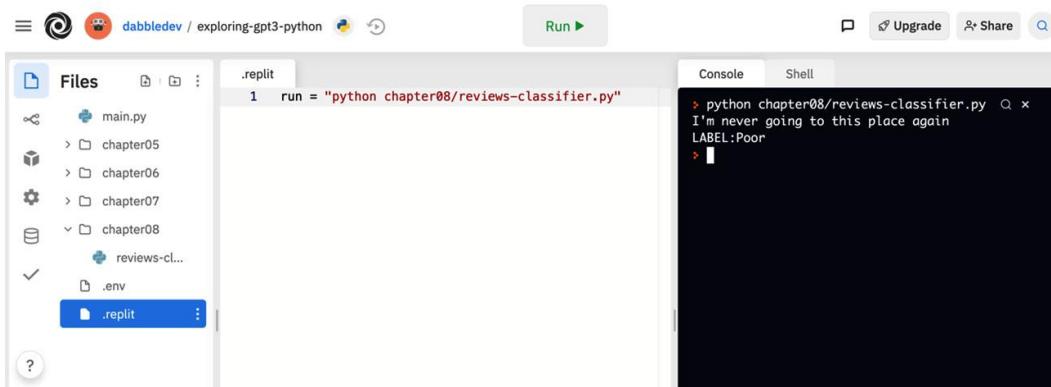


Figure 8.2 – Example output from chapter08/online-review-classifier.py

The screenshot shows the Replit IDE interface. The left sidebar displays a file tree for the 'chapter08' directory, which contains 'index.js', 'chapter05', 'chapter06', 'chapter07', 'chapter08' (containing 'email-clas...', 'esrb-ratin...', 'keywords-...', 'reviews-cl...', and '.env'), and a '.replit' file. The main workspace shows a command-line interface with the following interaction:

```
1 run = "node chapter08/esrb-rating-classifier.js"
> node chapter08/esrb-rating-classifier.js
Provide an ESRB rating for the following text:
>i'm going to hunt you down, and when I find you, I'll make you wish you were dead.
ESRB rating: M, because of "violence, blood, suggestive themes, and strong language."
> |
```

Figure 8.3 – Example output from chapter08/esrb-rating-classifier.js

The screenshot shows the Replit IDE interface. The left sidebar displays a file tree for the 'chapter08' directory, which contains 'index.js', 'chapter05', 'chapter06', 'chapter07', 'chapter08' (containing 'esrb-r...', 'keyw...', 'langu...', 'revie...', and '.env'), and a '.replit' file. The main workspace shows a command-line interface with the following interaction:

```
1 run = "node chapter08/language-classifier.js"
> node chapter08/language-classifier.js
¿Con quién debo comunicarme sobre ofertas de trabajo técnico?
LABEL:Spanish
> |
```

Figure 8.5 – Example output from chapter08/language-classifier.js

The screenshot shows the Replit IDE interface. On the left, the file tree displays a project structure with files like main.py, chapter05, chapter06, chapter07, chapter08 (containing esrb-r..., keyw..., langu..., revie...), and a .replit file. The central editor window contains the following Python code:

```
1 run = "python
chapter08/language-classifier.py"
```

The right panel shows the console output:

```
> python chapter08/language-classifier.py
;Con quién debo comunicarme sobre ofertas de trabajo técnico?
LABEL:Spanish
```

Figure 8.6 – Example output from chapter08/language-classifier.py

The screenshot shows the Replit IDE interface. On the left, the file tree displays a project structure with files like index.js, chapter05, chapter06, chapter07, chapter08 (containing esrb-r..., keyw..., langu..., revie...), and a .replit file. The central editor window contains the following Node.js code:

```
1 run = "node
chapter08/keywords-classifier.js"
```

The right panel shows the console output, which is a large block of text about the history of NASA's space program, followed by the keyword summary:

```
t high profile program was Project Mercury, an effort to learn if humans could survive in space. This was followed by Project Gemini, which used spacecraft built for two astronauts to perfect the capabilities needed for the national objective of a human trip to the Moon by the end of the 1960s. Project Apollo achieved that objective in July 1969 with the Apollo 11 mission and expanded on it with five more successful lunar landing missions through 1972. After the Skylab and Apollo-Soyuz Test Projects of the mid-1970s, NASA's human spaceflight efforts again resumed in 1981, with the Space Shuttle program that continued for 30 years. The Shuttle was not only a breakthrough technology, but was essential to our next major step in space, the construction of the International Space Station.

Keywords: NASA, National Aeronautics and Space Administration, NASA, National Aeronautics and Space Administration, Human spaceflight, Human spaceflight
```

Figure 8.7 – Example output from chapter08/keywords-classifier.js

The screenshot shows the Replit IDE interface. On the left, the file tree displays a project structure with files like main.py, chapter05, chapter06, chapter07, chapter08 (containing esrb-r..., keyw..., langu..., revie...), and a .replit file. The central editor window contains the following Python code:

```
1 run = "python
chapter08/keywords-classifier.py"
```

The right panel shows the console output, which is identical to Figure 8.7, displaying the same historical text and keyword summary.

Figure 8.8 – Example output from chapter08/keywords-classifier.py

Chapter 9

Figures

GPT Answers

An Example Knowledge Base App Powered by GPT-3

What is your favorite food?	GET ANSWER
-----------------------------	------------

Carrot cake.

Figure 9.1 – GPT Answers user interface

The screenshot shows a terminal window with the following content:

```
Run ▶          ⌂ Upgrade ⌂ Share ⌂
Console Shell
~/gpt-cv-node$ npx express-generator --no-view --force .
npx: installed 10 in 5.774s

  create : public/
  create : public/javascripts/
  create : public/images/
  create : public/stylesheets/
  create : public/stylesheets/style.css
  create : routes/
  create : routes/index.js
  create : routes/users.js
  create : public/index.html
  create : app.js
  create : package.json
  create : bin/
  create : bin/www

  install dependencies:
    $ npm install

  run the app:
    $ DEBUG=gpt-cv-node:* npm start

~/gpt-cv-node$
```

Figure 9.2 – Output from express-generator

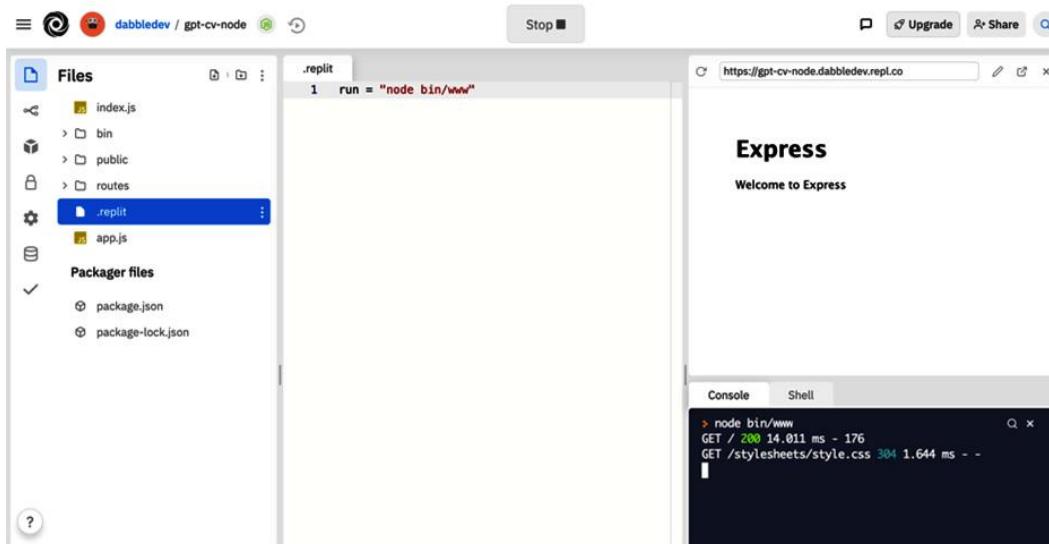
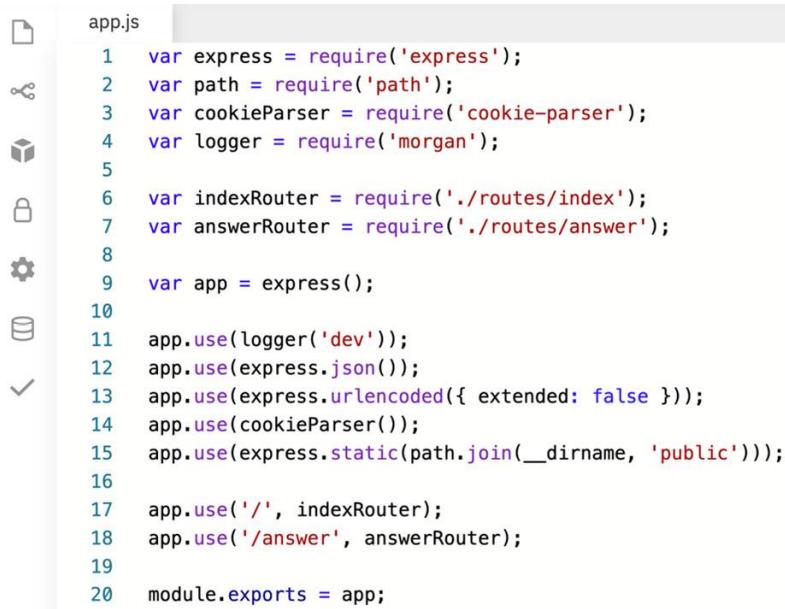


Figure 9.3 – Express server running in Replit.com

```
app.js
1 var express = require('express');
2 var path = require('path');
3 var cookieParser = require('cookie-parser');
4 var logger = require('morgan');
5
6 var indexRouter = require('./routes/index');
7 var usersRouter = require('./routes/users');
8
9 var app = express();
10
11 app.use(logger('dev'));
12 app.use(express.json());
13 app.use(express.urlencoded({ extended: false }));
14 app.use(cookieParser());
15 app.use(express.static(path.join(__dirname, 'public')));
16
17 app.use('/', indexRouter);
18 app.use('/users', usersRouter);
19
20 module.exports = app;
```

Figure 9.4 – Default app.js created by express-generator



```

app.js
1 var express = require('express');
2 var path = require('path');
3 var cookieParser = require('cookie-parser');
4 var logger = require('morgan');
5
6 var indexRouter = require('./routes/index');
7 var answerRouter = require('./routes/answer');
8
9 var app = express();
10
11 app.use(logger('dev'));
12 app.use(express.json());
13 app.use(express.urlencoded({ extended: false }));
14 app.use(cookieParser());
15 app.use(express.static(path.join(__dirname, 'public')));
16
17 app.use('/', indexRouter);
18 app.use('/answer', answerRouter);
19
20 module.exports = app;

```

Figure 9.5 – Edited app.js file

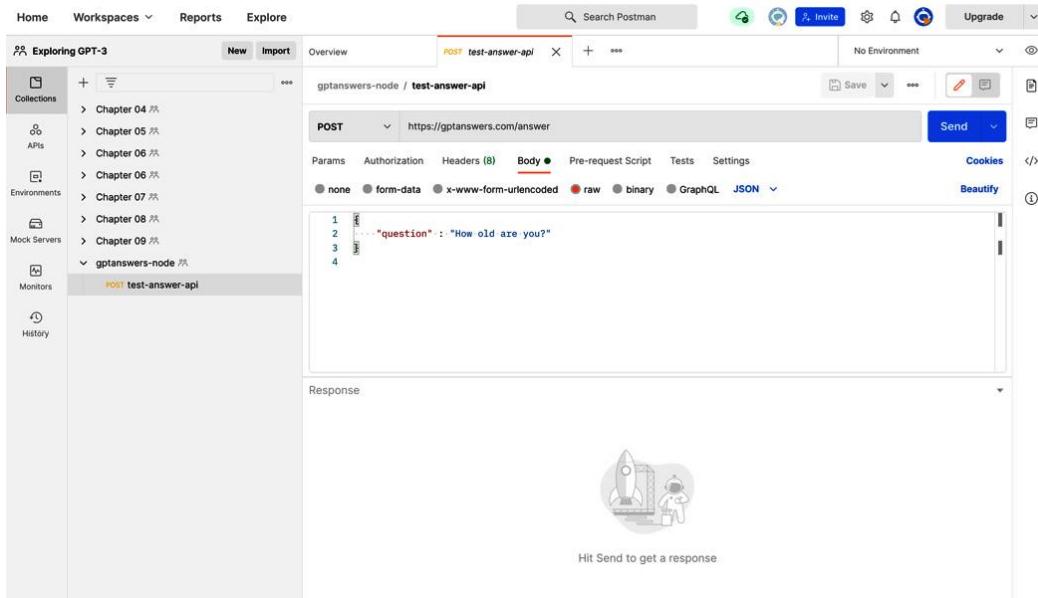


Figure 9.6 – Postman request to test the GPT-CV app API endpoint

```

public/index.html
1  <html>
2
3  <head>
4      <title>GPT Answers</title>
5      <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/css/uikit.min.css">
6  </head>
7
8  <body>
9
10 <div class="uk-section uk-section-large uk-height-viewport">
11     <div class="uk-container uk-text-center uk-padding-large">
12         <h1 class="uk-heading-medium"><strong>GPT</strong> Answers </h1>
13         <p class="uk-text-lead">An Example Knowledge Base App Powered by GPT-3</p>
14     </div>
15     <div class="uk-container uk-text-center">
16         <form class="uk-grid-collapse" uk-grid>
17             <div class="uk-width-1-1 ">
18                 <input id="question" class="uk-input uk-width-1-3" type="text">
19                 <button type="submit" class="uk-button uk-button-default uk-width-1-5">Get Answer</button>
20             </div>
21         </form>
22     </div>
23     <div class="uk-container uk-text-center uk-padding">
24         <div class="uk-inline">
25             <div id="answer" class="uk-flex uk-flex-center uk-flex-middle uk-padding uk-width-expand"></div>
26         </div>
27     </div>
28 </div>
29
30 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
31 <script src="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/js/uikit.min.js"></script>
32 <script src="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/js/uikit-icons.min.js"></script>
33 <script src="/js/script.js"></script>
34 </body>
35
36 </html>

```

Figure 9.7 – Completed index.html code

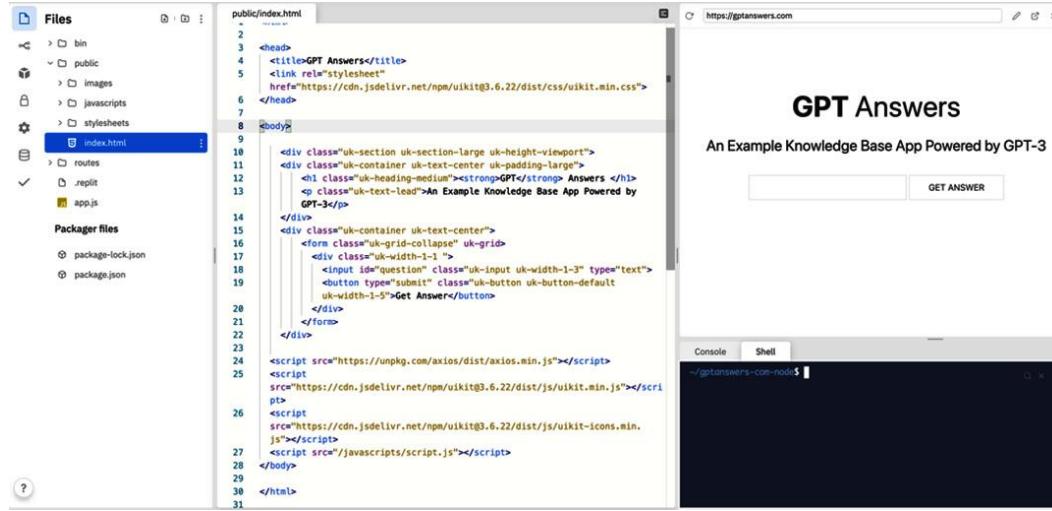


Figure 9.8 – Question input form

GPT Answers

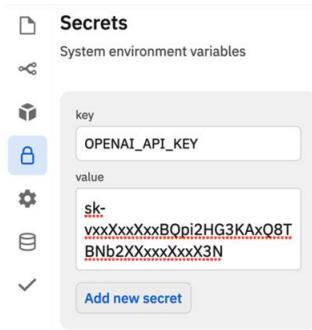
An Example Knowledge Base App Powered by GPT-3



Figure 9.9 – Testing the web UI with the placeholder API code

```
routes/answer.js
1  const axios = require('axios');
2  const express = require('express');
3  const router = express.Router();
4
5  const apiKey = process.env.OPENAI_API_KEY;
6  const client = axios.create({
7    headers: { 'Authorization': 'Bearer ' + apiKey }
8  });
9
10 const documents = [
11   "I am a day older than I was yesterday.<|endoftext|>",
12   "I build applications that use GPT-3.<|endoftext|>",
13   "My preferred programming is Python.<|endoftext|>"
14 ]
15
16 const endpoint = 'https://api.openai.com/v1/answers';
17
18 router.post('/', (req, res) => {
19   const data = {
20     "file": process.env.ANSWERS_FILE,
21     "question": req.body.question,
22     "search_model": "ada",
23     "model": "curie",
24     "examples_context": "My favorite programming language is Python.",
25     "examples": [{"How old are you?", "I'm a day older than I was yesterday."}, {"What languages do you know?", "I speak English and write code in Python."}],
26     "max_tokens": 15,
27     "temperature": 0,
28     "return_prompt": false,
29     "expand": ["completion"],
30     "stop": ["\n", "<|endoftext|>"]
31   }
32   client.post(endpoint, data)
33     .then(result => {
34       res.send({"answer" : result.data.answers[0]})
35     }).catch(result => {
36       res.send({"answer" : "Sorry, I don't have an answer."})
37     });
38 });
39
40 module.exports = router;
41
42
```

Figure 9.10 – Edited routes/answer.js file



```
routes/answer.js
1 const axios = require('axios');
2 const express = require('express');
3 const router = express.Router();
4
5 const apiKey = process.env.OPENAI_API_KEY;
6 const client = axios.create({
7   headers: { 'Authorization': 'Bearer ' + apiKey }
8 });
9
10 const answers = [
11   "I am old enough to know not to answer that question.<|endoftext|>",
12   "I write books and create video courses to help people learn about GPT-3.<|endoftext|>",
13   "My preferred programming languages are C++, C#, JavaScript, Go, and Python.<|endoftext|>"
14 ]
15
16 const endpoint = 'https://api.openai.com/v1/answers';
17
18 router.post('/', (req, res) => {
19   const data = {
20     "documents": answers,
21     "question": req.body.question || "what is this?",
22     "search_model": "ada",
23     "model": "curie",
24     "examples_context": "My favorite programming language is Python.",
25     "examples": [{"How old are you?", "I'm a day older than I was yesterday."}]
26   }
27   res.json(data);
28 });
29
30 module.exports = router;
```

Figure 9.11 – Add a secret for your OpenAI API key

GPT Answers

An Example Knowledge Base App Powered by GPT-3

What is your favorite food?

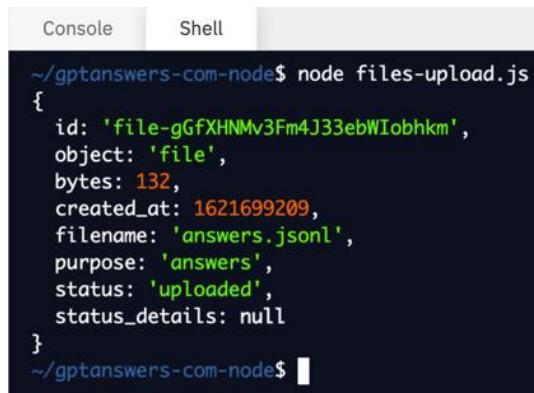
GET ANSWER

I like pizza.

Figure 9.12 – Answer from GPT-3

```
files-upload.js
 2  const axios = require('axios');
 3  const FormData = require('form-data');
 4
 5  const data = new FormData();
 6  data.append('purpose', 'answers');
 7  data.append('file', fs.createReadStream('answers.jsonl'));
 8
 9  const params = {
10    method: 'post',
11    url: 'https://api.openai.com/v1/files',
12    headers: {
13      'Authorization': 'Bearer ' + process.env.OPENAI_API_KEY,
14      ...data.getHeaders()
15    },
16    data: data
17  }
18
19  axios(params)
20    .then(function(response) {
21      console.log(response.data);
22    })
23    .catch(function(error) {
24      console.log(error.message);
25    });
26
```

Figure 9.13 – Completed code for file-upload.js

A screenshot of a terminal window titled "Console". The window shows the command "node files-upload.js" being run, followed by its JSON output. The output is a single object with the following properties:

```
~/gptanswers-com-node$ node files-upload.js
{
  id: 'file-gGfXHNMv3Fm4J33ebWIobhkm',
  object: 'file',
  bytes: 132,
  created_at: 1621699209,
  filename: 'answers.jsonl',
  purpose: 'answers',
  status: 'uploaded',
  status_details: null
}
```

Figure 9.14 – Shell output from files-upload.js

```
18 router.post('/', (req, res) => {
19   const data = {
20     "file": process.env.ANSWERS_FILE,
21     "question": req.body.question,
22     "search_model": "ada",
23     "model": "curie",
24     "examples_context": "My favorite programming language is Python.",
25     "examples": [{"How old are you?", "I'm a day older than I was yesterday."},
26                  {"What languages do you know?", "I speak English and write code in Python."}],
27     "max_tokens": 15,
28     "temperature": 0,
29     "return_prompt": false,
30     "expand": ["completion"],
31     "stop": ["\n", "<|endoftext|>"]
32   }
33 }
```

Figure 9.15 – The Answers endpoint parameters using the file parameter

GPT Answers

An Example Knowledge Base App Powered by GPT-3

What is your favorite food?

GET ANSWER

Carrot cake.

Figure 9.16 – An answer generated from the answers file

GPT Answers

An Example Knowledge Base App Powered by GPT-3

Do you sell this one in red?

GET ANSWER

Sorry, I don't have an answer.

Figure 9.17 – A question that can't be answered

GPT Answers

An Example Knowledge Base App Powered by GPT-3

What is your favorite vacation spot?	GET ANSWER
--------------------------------------	------------

I like to go to the beach.

Figure 9.18 – An answer that isn't from the answers file

Links

- The URL for the Answers endpoint is <https://api.openai.com/v1/answers>
- see the OpenAI docs for the Answers endpoint located at <https://beta.openai.com/docs/api-reference/answers>
- <https://getuikit.com/>

Code

Code 9.1 – In the output pane, click on the Shell tab and enter this command:

```
npx express-generator --no-view --force
```

Code 9.2 - After `express-generator` completes, run the following command in the shell:

```
npm update
```

Code 9.3 - Now create a file named `.replit` and add the following `Run` command to it:

```
Run = "node ./bin/www"
```

Code 9.4 – Edit `line 7` and change `var usersRouter = require('./routes/users')` to the following:

```
var answerRouter = require('./routes/answer');
```

Code 9.5 – Edit `line 18` and change `app.use('/users', usersRouter);` to the following:

```
app.use('/answer', answerRouter);
```

Code 9.6 - Add the following code to the `answers.js` file:

```
const axios = require('axios');

const express = require('express');
const router = express.Router();

router.post('/', (req, res) => {
  res.send({answer:'placeholder for the answer'});
});

module.exports = router;
```

Code 9.7 – The format of endpoint URL

```
https://gptanswers-node.{username}.repl.co
```

Code 9.8 – Finally, add the following JSON for the request body:

```
{  
  "question" : "How old are you?"  
}
```

Code 9.9 – Click the blue **Send** button to submit the request and review the response, which should be the following:

```
{  
  "answer": "placeholder for the answer"  
}
```

Code 9.10 – Replace the URL for the style sheet with

```
https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/css/uikit.min.css.
```

Code 9.11 – Replace everything between the `<body>` tag and the `</body>` tag with the following:

```
<div class="uk-section uk-section-large uk-height-viewport">
```

```

<div class="uk-container uk-text-center uk-padding-large">

    <h1 class="uk-heading-medium"><strong>GPT</strong> Answers
</h1>

    <p class="uk-text-lead">An Example Knowledge Base App Powered
by GPT-3</p>

</div>

<div class="uk-container uk-text-center">

    <form class="uk-grid-collapse" uk-grid>

        <div class="uk-width-1-1 ">
            <input id="question" class="uk-input uk-width-1-3"
type="text">

            <button type="submit" class="uk-button uk-button-default
uk-width-1-5">Get Answer</button>
        </div>
    </form>
</div>

<div class="uk-container uk-text-center uk-padding">

    <div class="uk-inline">
        <div id="answer" class="uk-flex uk-flex-center uk-flex-middle
uk-padding uk-width-expand"></div>
    </div>
</div>
</div>

```

Code 9.12 – Add code 9.12 above the `</body>` tag.

```

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/js/uikit.min.js"><
/script>
<script src="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/js/uikit-
icons.min.js"></script>
<script src="/javascripts/script.js"></script>

```

Code 9.13 – First, we'll add two variables to hold the HTML form and answer `div`:

```
const form = document.querySelector('form');  
const answer = document.querySelector('#answer');
```

Code 9.14 – Next, we'll add code that will fire when the form is submitted:

```
const formEvent = form.addEventListener('submit', event => {  
  event.preventDefault();  
  
  const question = document.querySelector('#question');  
  
  if (question.value) {  
    askQuestion(question.value);  
  } else {  
  
    answer.innerHTML = "You need to enter a question to get an  
    answer.";  
  
    answer.classList.add("error");  
  }  
});
```

Code 9.15 – The following code will append the text to the answer `div`:

```
const appendAnswer = (result) => {  
  
  answer.innerHTML = `<p>${result.answer}</p>`;  
};
```

Code 9.16 – Finally, we'll add a function to call the app API endpoint using Axios:

```
const askQuestion = (question) => {  
  
  const params = {  
  
    method: 'post',  
  
    url: '/answer',  
  
    headers: {  
  
      'content-type': 'application/json'  
    },  
  
    data: { question }  
};
```

```

};

axios(params)

.then(response => {
  const answer = response.data;
  appendAnswer(answer);
})

.catch(error => console.error(error));
};

```

Code 9.17 – Starting on **line 5**, add the following code followed by a line return:

```

const apiKey = process.env.OPENAI_API_KEY;

const client = axios.create({
  headers: { 'Authorization': 'Bearer ' + apiKey }
});

```

Code 9.18 – Next, add the following code with a line return after it:

```

const documents = [
  "I am a day older than I was yesterday.<|endoftext|>",
  "I build applications that use GPT-3.<|endoftext|>",
  "My preferred programming is Python.<|endoftext|>"
]

```

Code 9.19 – Add the following code starting on **line 16** followed by a line return:

```
const endpoint = 'https://api.openai.com/v1/answers';
```

Code 9.20 – Next, beginning on **line 18**, add the following to complete the code:

```

router.post('/', (req, res) => {
  // call the OpenAI API
  const data = {
    "documents": documents,
}

```

```

    "question": req.body.question,
    "search_model": "ada",
    "model": "curie",
    "examples_context": "My favorite programming language is Python.",
    "examples": [["How old are you?", "I'm a day older than I was yesterday."], ["What languages do you know?", "I speak English and write code in Python."]],
    "max_tokens": 15,
    "temperature": 0,
    "return_prompt": false,
    "expand": ["completion"],
    "stop": ["\n", "<|endoftext|>"],
  }
  client.post(endpoint, data)
  .then(result => {
    res.send({"answer" : result.data.answers[0] })
  }).catch(result => {
    res.send({"answer" : "Sorry, I don't have an answer."})
  });
}

module.exports = router;

```

Code 9.21 – Create a new `json` file named `answers.json` and some answers for the file in the following format:

```

{"text": "I am a day younger than I will be tomorrow"}
{"text": "I like to code in Python."}
{"text": "My favorite food is carrot cake."}

```

Code 9.22 – Create another new file named `files-upload.js` and add the following code to it. Require a few modules that will be used:

```

const fs = require('fs');

const axios = require('axios');

```

```
const FormData = require('form-data');
```

Code 9.23 – Next, add the following code to read in the `jsonl` data for the request:

```
const data = new FormData();

data.append('purpose', 'answers');

data.append('file', fs.createReadStream('answers.jsonl'));
```

Code 9.24 – Add a variable for the HTTP request parameters:

```
const params = {

method: 'post',
url: 'https://api.openai.com/v1/files',
headers: {
  'Authorization': 'Bearer ' + process.env.OPENAI_API_KEY,
  ...data.getHeaders()
},
data : data
}
```

Code 9.25 – Finally, add code to make the HTTP request and log results:

```
axios(params)

.then(function(response) {
  console.log(response.data);
})

.catch(function(error) {
  console.log(error);
});
```

Code 9.25 – At the `~/gptanswers-node` prompt, enter the following command with your OpenAI API key:

```
export OPENAI_API_KEY="your-api-key-goes-here"
```

Code 9.26 – Next, enter the following command in the shell:

```
node files-upload.js
```

Code 9.27 – Run the following shell command to set your API key as an environment variable that the shell can access:

```
export OPENAI_API_KEY="your-api-key-goes-here"
```

Code 9.28 – Run the following command in the shell to execute `files-upload.js`:

```
node files-upload.js
```

Chapter 10

Links

- <https://beta.openai.com/docs/use-case-guidelines>
- <https://beta.openai.com/docs/safety-best-practices>
- <https://beta.openai.com/forms/pre-launch-review>
- <https://beta.openai.com/forms/quota-increase>

Code

Code 10.1 – Require the bad-words library on the first line of routes/answer.js with the following code:

```
const Filter = require('bad-words');
```

Code 10.2 – In the `routes/answer.js` file, add the following code above the line that begins with `const data`:

```
let filter = new Filter();

if (filter.isProfane(req.body.question)) {
    res.send({ "answer": "That's not a question we can answer." });
    return;
}
```

Code 10.3 - Open `routes/answer-dot-JS` and add the following code on a new line after the line that begins with `router.post`:

```
if (req.body.question.length > 150) {
    res.send({ "answer": "Sorry. That question is too long." });
    return;
}
```

Code 10.4 – Open `app.js` and after `line 9` (or after `var app = express();`), add the following code:

```
const rateLimit = require("express-rate-limit");
```

```
const apiLimiter = rateLimit({  
  windowMs: 1 * 60 * 1000,  
  max: 6  
});  
  
app.use("/answer/", apiLimiter);
```

Code 10.5 – Open `routes/answer-dot-JS` and add code 10.5 after the line that begins with `router.post:`

```
if (req.rateLimit.remaining == 0) {  
  res.send({"answer" : "Ask me again in a minute."});  
  return;  
};
```

Figures

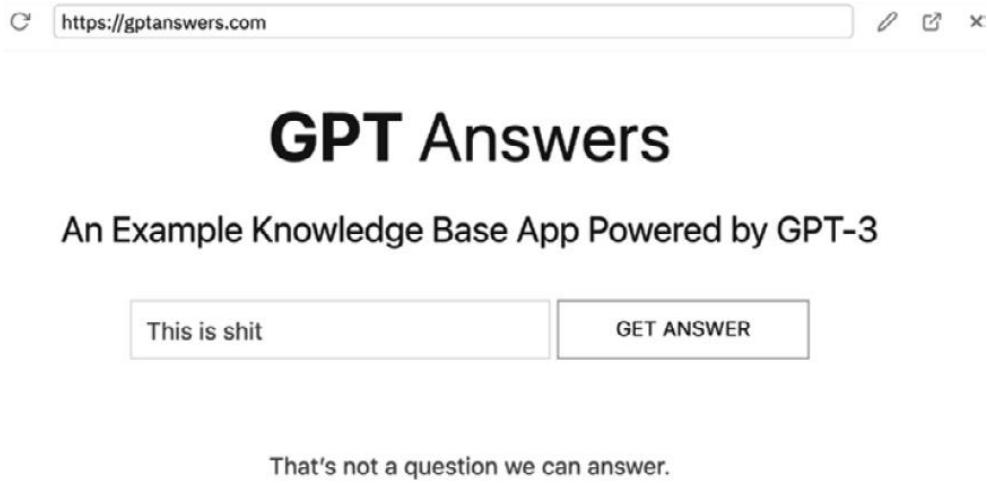
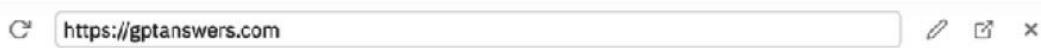


Figure 10.1 – Filtering profanity in questions



GPT Answers

An Example Knowledge Base App Powered by GPT-3

An injection attack is an attack that

GET ANSWER

Sorry. That question is too long.

Figure 10.2 – Form output with long text



GPT Answers

An Example Knowledge Base App Powered by GPT-3

What is your favorite food?

GET ANSWER

Ask me again in a minute.

Figure 10.3 – Message when request rate is exceeded

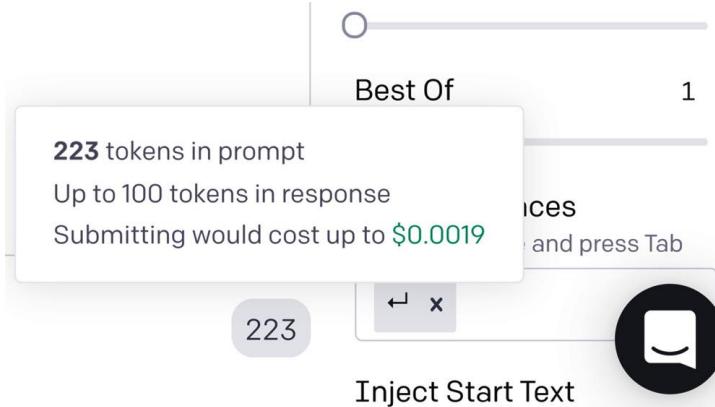


Figure 10.4 – Estimated cost