**Generating and transforming text**

**WORKING WITH THE OPENAI API**

# James Chapman

Curriculum Manager, DataCamp

**Recap...**

**Q&A**

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

"How many days are in October?"

}]

)

print

(

response.choices

[

0

]

.message.content

)

October has 31 days.

**How is the output generated?**

Text most likely to complete the prompt

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

"Life is like a box of chocolates."

}]

)

print

(

response.choices

[

0

]

.message.content

)

You never know what you're going to get. This famous quote from the movie

"Forrest Gump"...

Response is **non-deterministic** (inherently random)

**Controlling response randomness**

temperature

:

control on

d

e

t

e

r

m

i

n

i

s

m

Ranges from

0

(highly deterministic) to

2

(very random)

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

"Life is like a box of chocolates."

}]

,

temperature=

2

)

print

(

response.choices

[

0

]

.message.content

)

"...you never know what you're gonna get." That quote reminds us of the unpredictability of life and the diverse set of experiences we might encounter. Whether sweet, nutty, bitter, or flashy, life holds a treasure trove of surprises.

# Content transformation

Changing text based on an instruction

Find and replace

Summarization

Copyediting

prompt =

"""

Update name to Maarten, pronouns to he/him, and job title to Senior Content Developer

in the following text:

Joanne is a Content Developer at DataCamp. Her favorite programming language is R,

which she uses for her statistical analyses.

"""

# Content transformation

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

prompt

}]

)

print

(

response.choices

[

0

]

.message.content

)

Maarten is a Senior Content Developer at DataCamp. His favorite programming language is R, which he uses for his statistical analyses.

# Content generation

prompt =

"Create a tagline for a new hot dog stand."

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

prompt

}]

:

)

print

(

response.choices

[

0

]

.message.content

)

"Frankly, we've got the BEST dogs in town!"

# Controlling response length

## Default max\_tokens

response = client.chat.completions.create( model="gpt-4o-mini", prompt="Write a haiku about AI."

) print(response.choices[0].text)

AI so powerful

Computers that think and learn

Superseding

# Controlling response length

## Default max\_tokens max\_tokens = 30

response = client.chat.completions.create( model="gpt-4o-mini", prompt="Write a haiku about AI."

) print(response.choices[0].text)

AI so powerful

Computers that think and learn

Superseding

response = client.chat.completions.create( model="gpt-4o-mini", prompt="Write a haiku about AI.", max\_tokens=30

) print(response.choices[0].text)

A machine mind thinks

Logic dictates its choices

Mankind ponders anew

# Understanding tokens



# Returning to cost

Usage costs dependent on amount of input and output text

Models are priced by cost/tokens

Input and output tokens can be priced differently

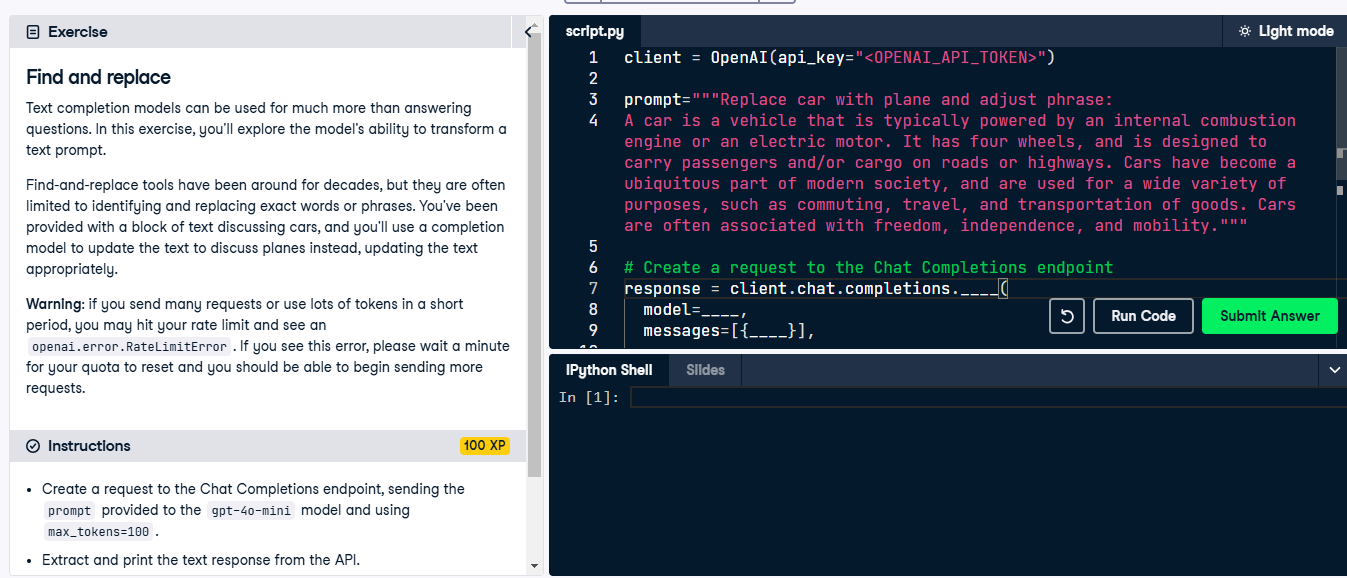
Increasing max\_tokens increases cost

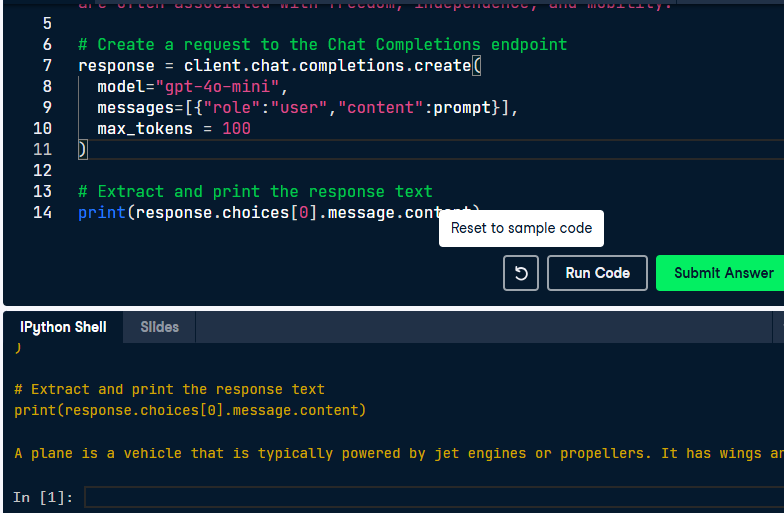
Scoping feature cost often starts with a rough calculation:

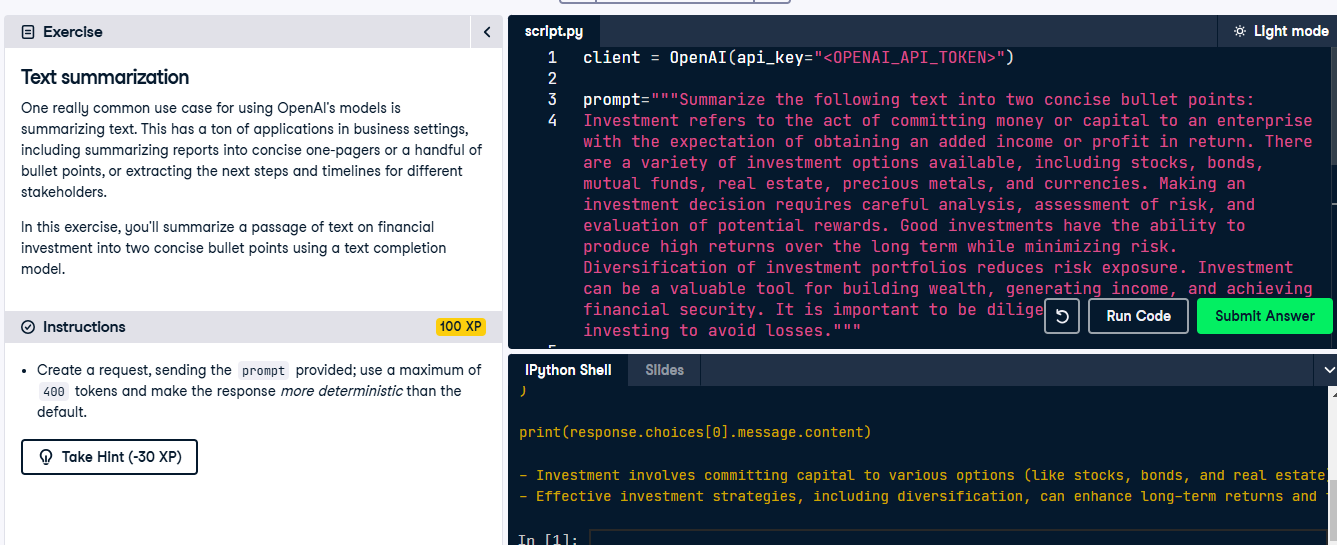
TimeCost = Avg. Tokens Generated × Model Cost × [[1]](#footnote-1)000 × Expected no. of requestsTime

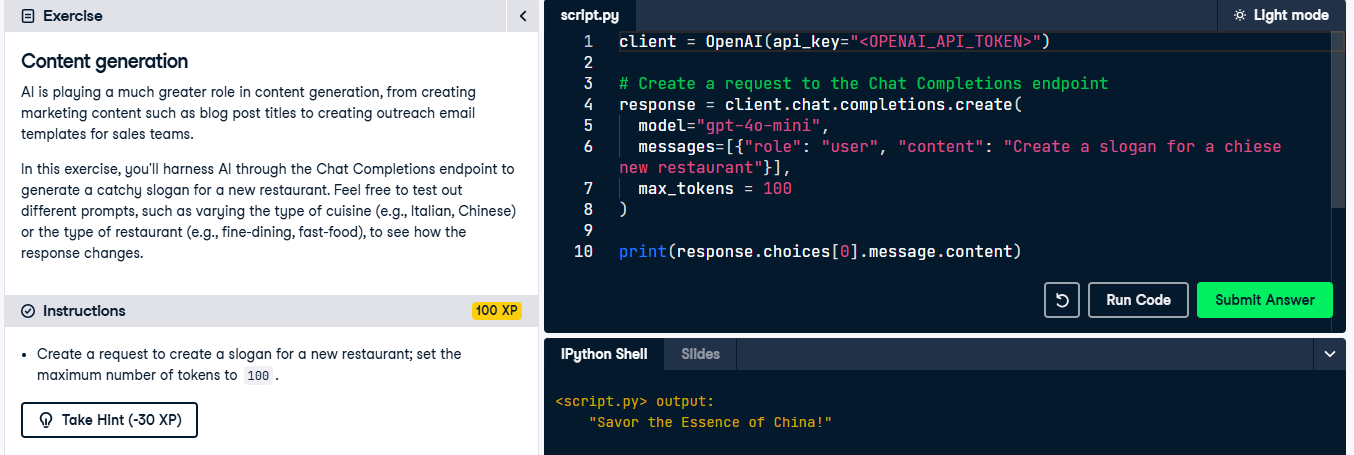
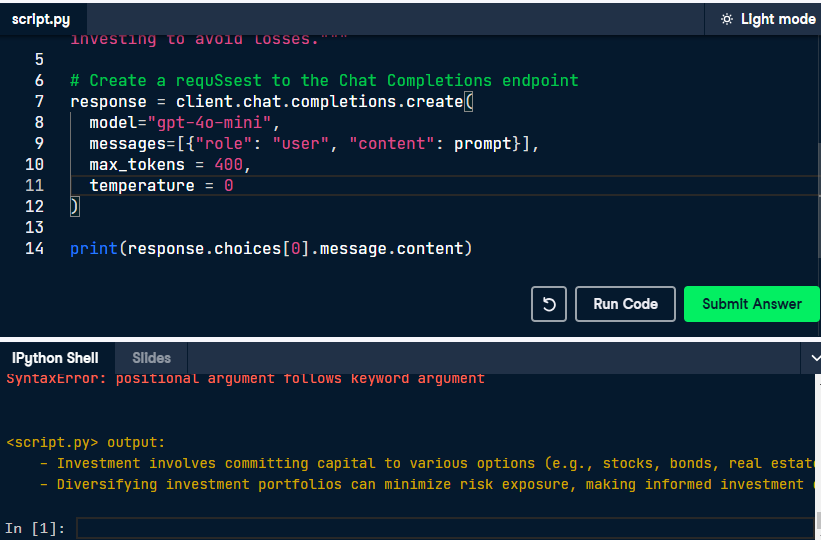
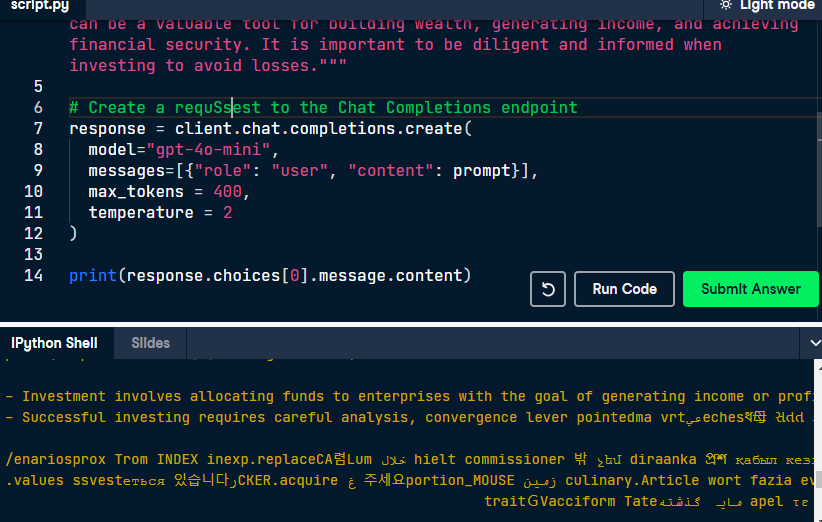
**Let's practice!**

**WORKING WITH THE OPENAI API**









**Sentiment analysis and classification**

**WORKING WITH THE OPENAI API**

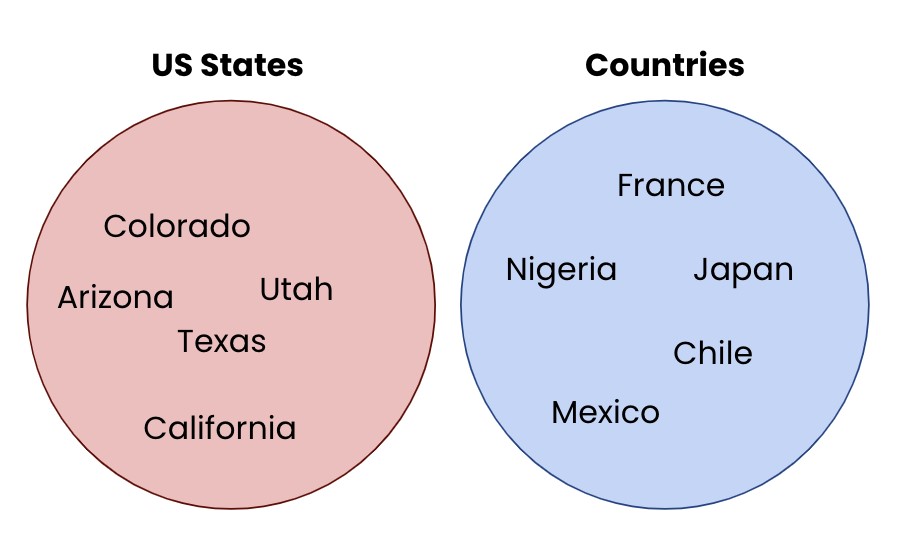
## James Chapman

Curriculum Manager, DataCamp

# Classification tasks

Task that involves assigning a label to

information

 Identifying the language from text

Categorization

Classifying sentiment

OpenAI models can perform these tasks, providing:

Model has sufficient knowledge

Prompt contains sufficient context

# Categorizing animals

response = client.chat.completions.create( model="gpt-4o-mini",

print

(

response.choices

[

0

]

.message.content

)

Here are the animals classified into categories based on their general classifications:

Mammals: Zebra, Polar Bear, Dog

Fish: Salmon

Reptiles: Crocodile

messages=[{"role": "user", "content": "Classify the following animals into categories: zebra, crocodile, blue whale, polar bear, salmon, dog."}], max\_tokens=50

)

# Specifying groups

Sure! Here's the classification of the animals you provided:

Animals with fur: Dog, Polar Bear, Zebra

Animals without fur: Crocodile, Dolphin, Salmon

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

"Classify the following animals into animals

with fur and without: zebra, crocodile, dolphin, polar bear, salmon, dog."

}]

,

max\_tokens=

50

)

print

(

response.choices

[

0

]

.message.content

)

prompt =

"""Classify sentiment in the following statements:

1

. The service was very slow

2

. The steak was awfully tasty

!

3

. Meal was decent, but I've had better.

4

. My food was delayed, but drinks were good.

"""

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

prompt}],

max\_tokens=

50

)

print

(

response.choices

[

0

]

.message.content

)

1. The service was very slow
2. The steak was awfully tasty!
3. Meal was decent, but I've had better.
4. My food was delayed, but drinks were good.

1

. Negative

2

. Positive

3

. Neutral

4

. Mixed

prompt =

"""Classify sentiment as 1-5 (bad-good) in the following statements:

1

. The service was very slow

2

. The steak was awfully tasty

!

3

. Meal was decent, but I've had better.

4

. My food was delayed, but drinks were good.

"""

1

.

1

2

.

5

3

.

3

4

.

2

# Zero-shot vs. one-shot vs. few-shot prompting

**Zero-shot** prompting: no examples provided

**In-context learning**:

**One-shot** prompting: one example provided

**Few-shot** prompting: multiple examples provided

# One-shot prompting

Neutral to Discontented

prompt =

"""Classify sentiment in the following statements:

The service was very slow // Disgruntled

Meal was decent, but I've had better. //

"""

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

prompt

}]

)

print

(

response.choices

[

0

]

.message.content

)

# Few-shot prompting

Neutral/Discontented

prompt =

"""Classify sentiment in the following statements:

The service was very slow // Disgruntled

The steak was awfully tasty! // Delighted

Good experience overall. // Satisfied

Meal was decent, but I've had better. //

"""

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

prompt

}]

)

print

(

response.choices

[

0

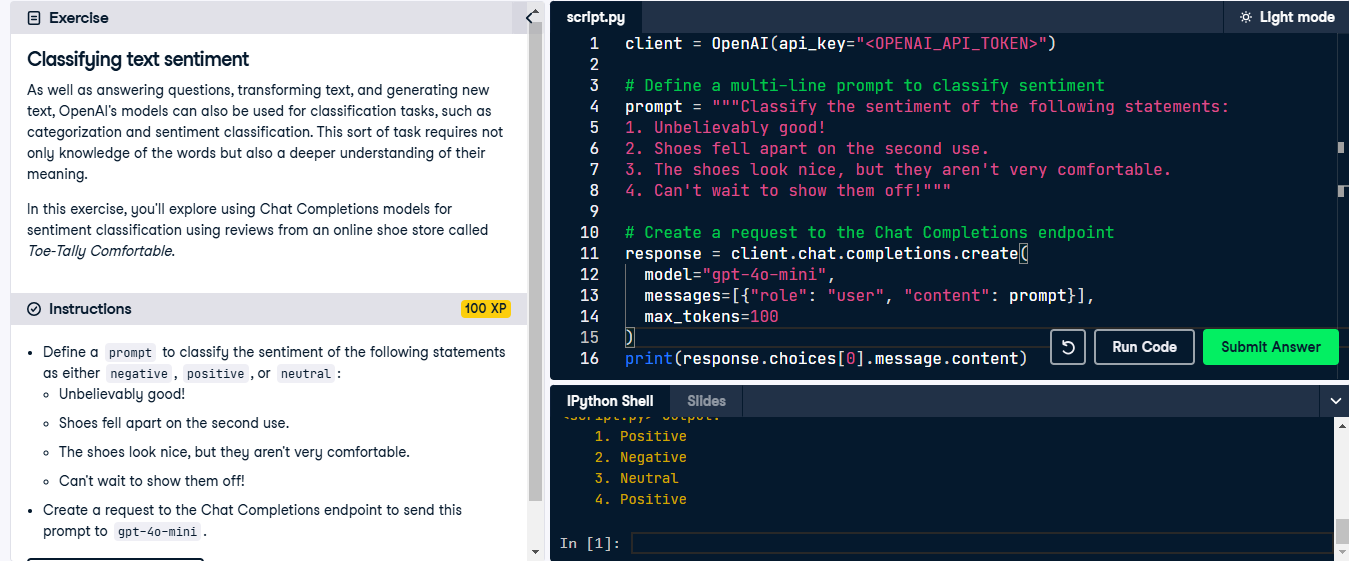
]

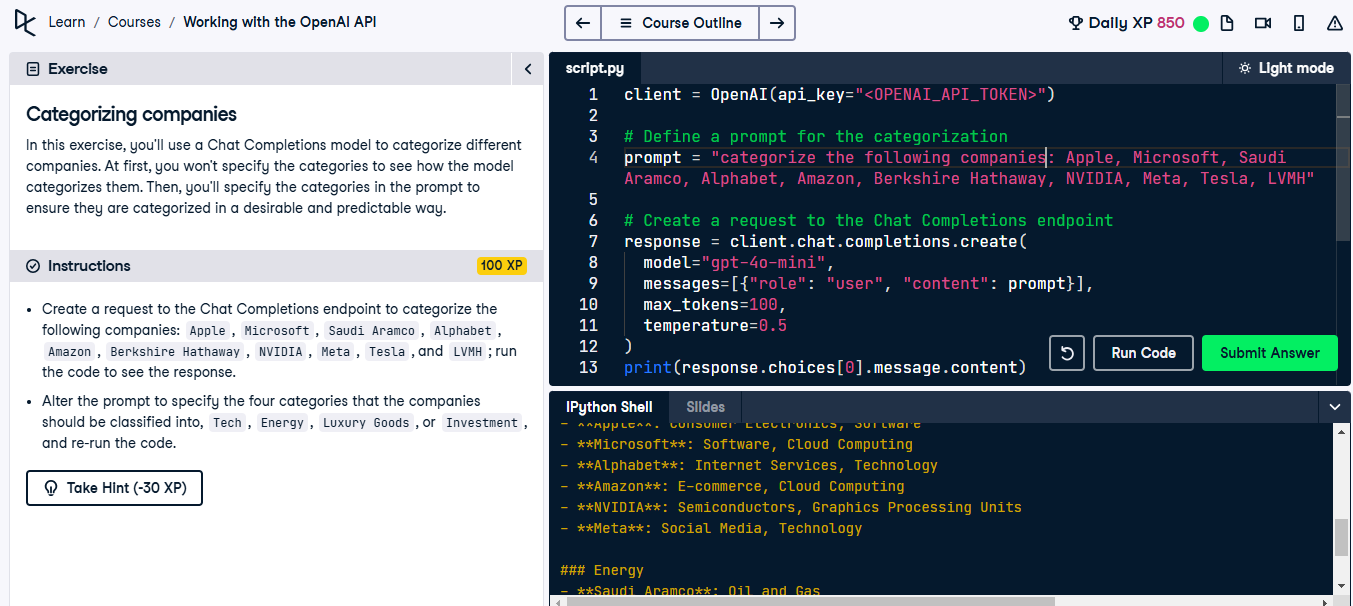
.message.content

)

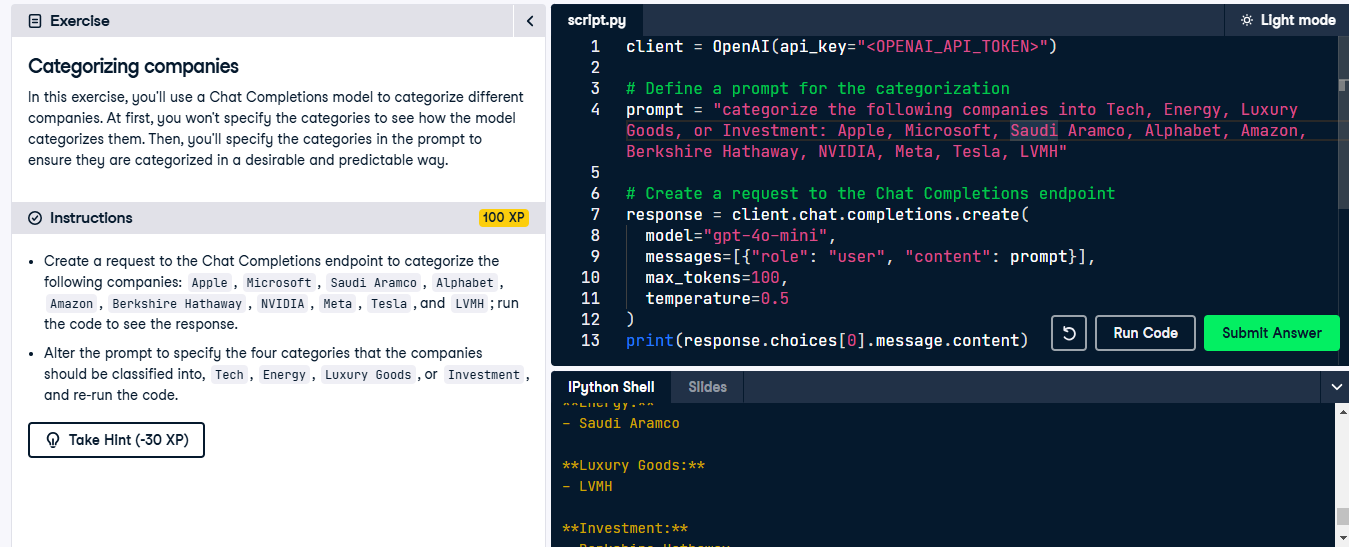
**Let's practice!**

**WORKING WITH THE OPENAI API**





Change the prompt



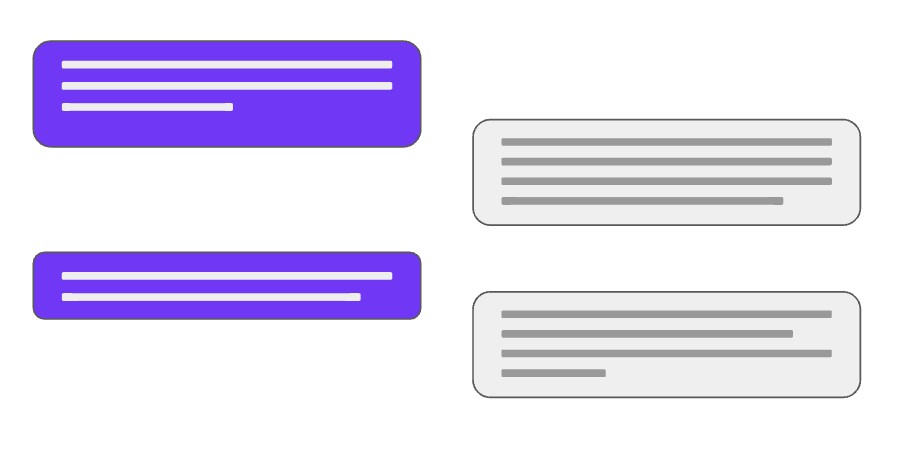
**Chat completions with GPT**

**WORKING WITH THE OPENAI API**

## James Chapman

Curriculum Manager, DataCamp

# The Chat Completions endpoint

Single-turn tasks

Text generation

Text transformation

Classification

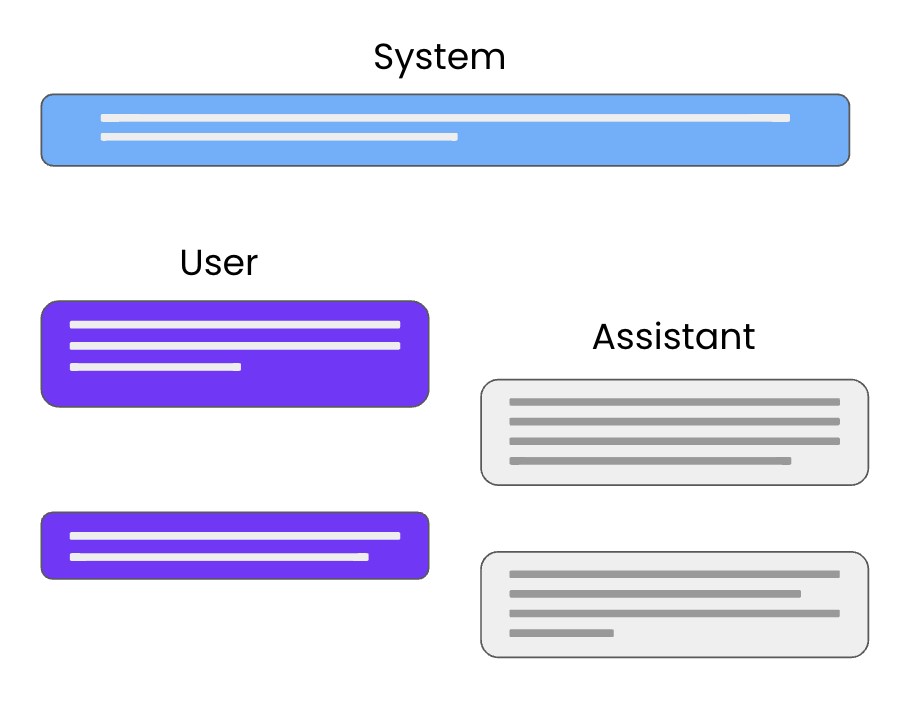
Multi-turn conversations

→ Build on previous prompts and responses

# Roles

**System**: controls assistant's behavior

**User**: instruct the assistant

 **Assistant**: response to user instruction Can also be written by the user to provide examples

# Request setup

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"user"

,

"content"

:

prompt

}]

)

# Prompt setup

messages=[{"role": "system",

"content": "You are a data science tutor who speaks concisely."},

{"role": "user",

"content": "What is the difference between mutable and immutable objects?"}]

# Making a request

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"system"

,

"content"

:

"You are a data science tutor who speaks concisely."

}

,

{

"role"

:

"user"

,

"content"

:

"What is the difference between mutable and immutable objects?"

}]

)

print

(

response.choices

[

0

]

.message.content

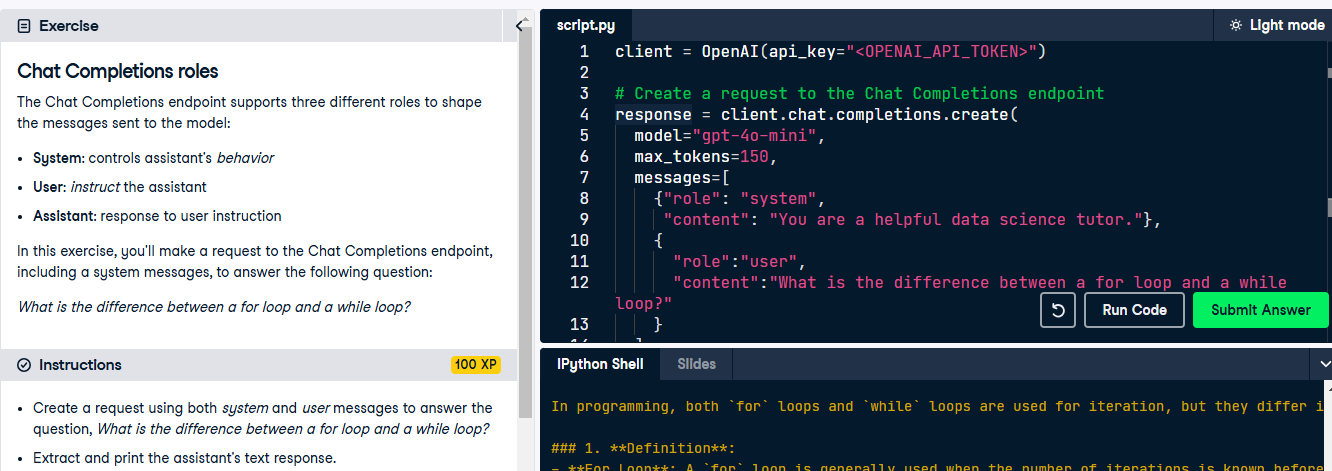
)

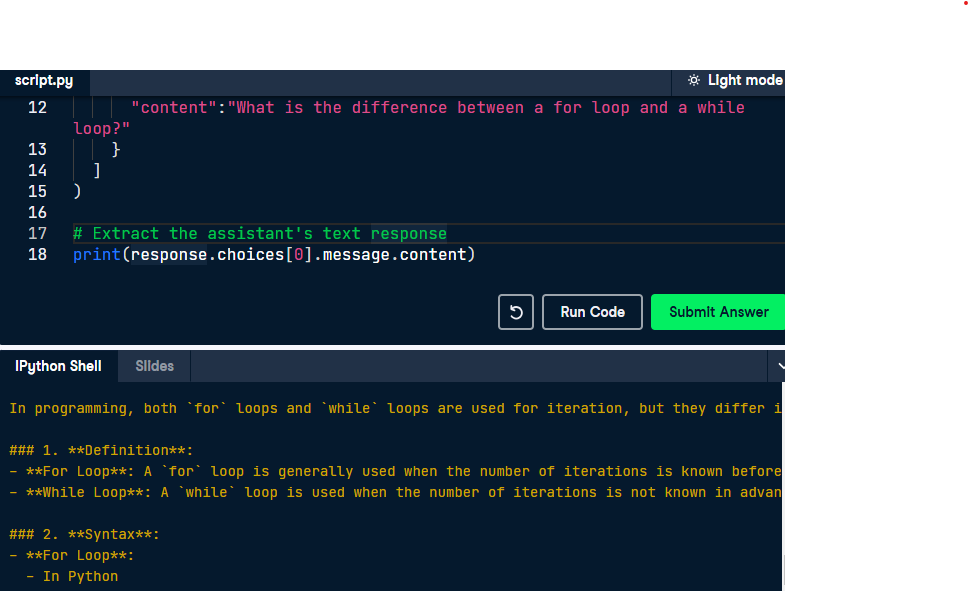
# The response

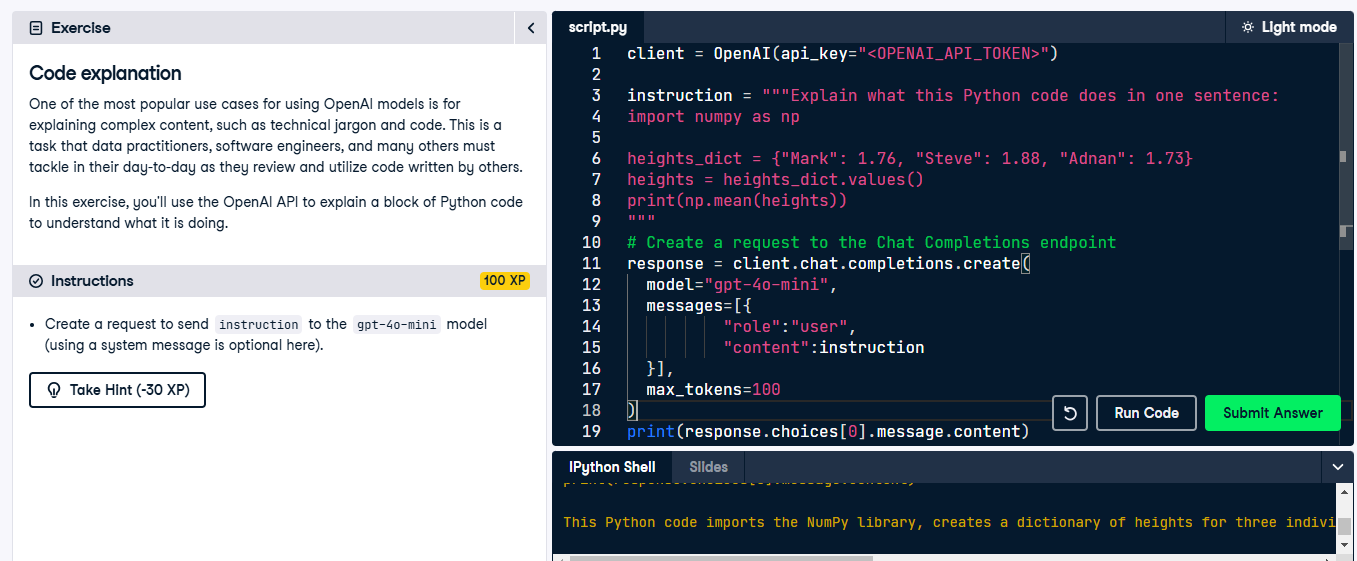
Mutable objects can be changed after creation, while immutable objects cannot be modified once they are created.

**Let's practice!**

**WORKING WITH THE OPENAI API**

****

****

****

**Multi-turn chat completions with GPT**

**WORKING WITH THE OPENAI API**

## James Chapman

Curriculum Manager, DataCamp

# Chat completions for single-turn tasks

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=[{

"role"

:

"system"

,

"content"

:

"You are a data science tutor."

}

,

{

"role"

:

"user"

,

"content"

:

"What is the difference between mutable and immutable objects?"

}]

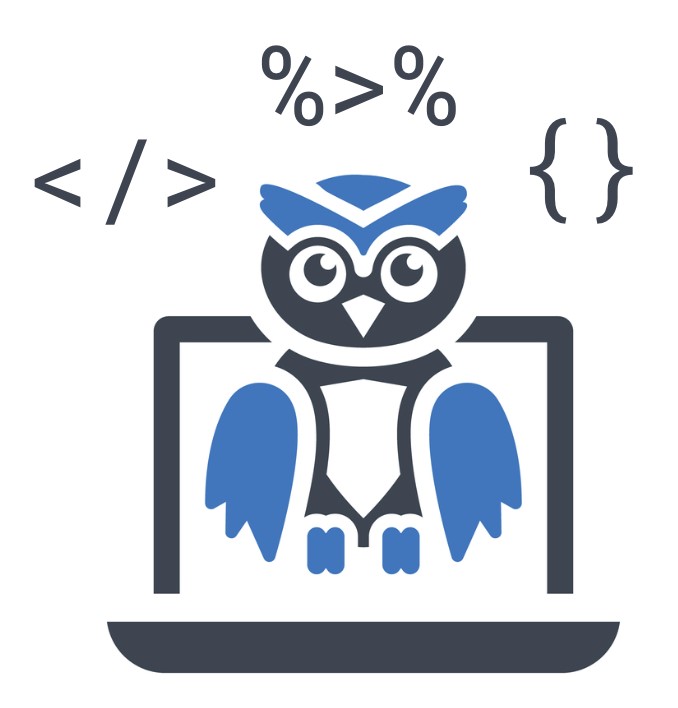
)

**System**: controls assistant's behavior

**User**: instruct the assistant

**Assistant**: response to user instruction

# Providing examples



Steer model in the right direction

Nothing surfaced to the end-user

**Example**: Data Science Tutor Application Provide examples of data science questions and answers

# Providing examples

response = client.chat.completions.create( model="gpt-4o-mini", messages=[{"role": "system",

"content": "You are a data science tutor who speaks concisely."},

{"role": "user",

"content": "How do you define a Python list?"},

{"role": "assistant",

"content": "Lists are defined by enclosing a comma-separated sequence of objects inside square brackets [ ]."},

{"role": "user",

"content": "What is the difference between mutable and immutable objects?"}] )

# The response

print

(

response.choices

[

0

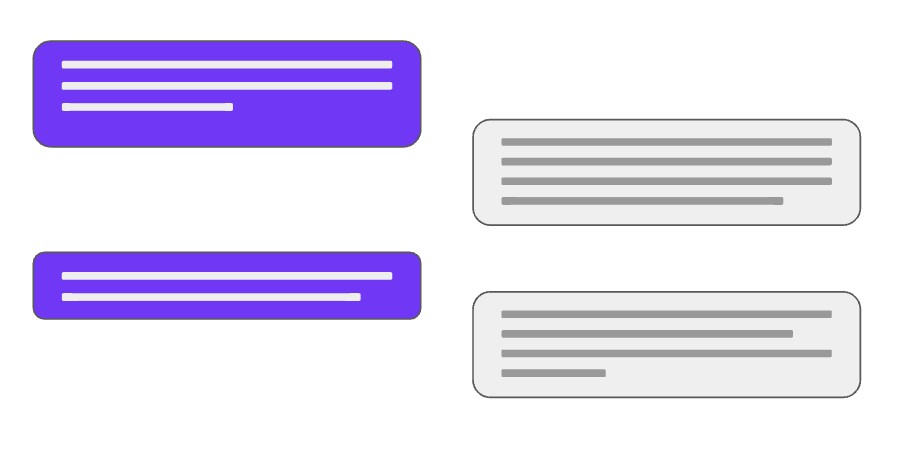
]

.message.content

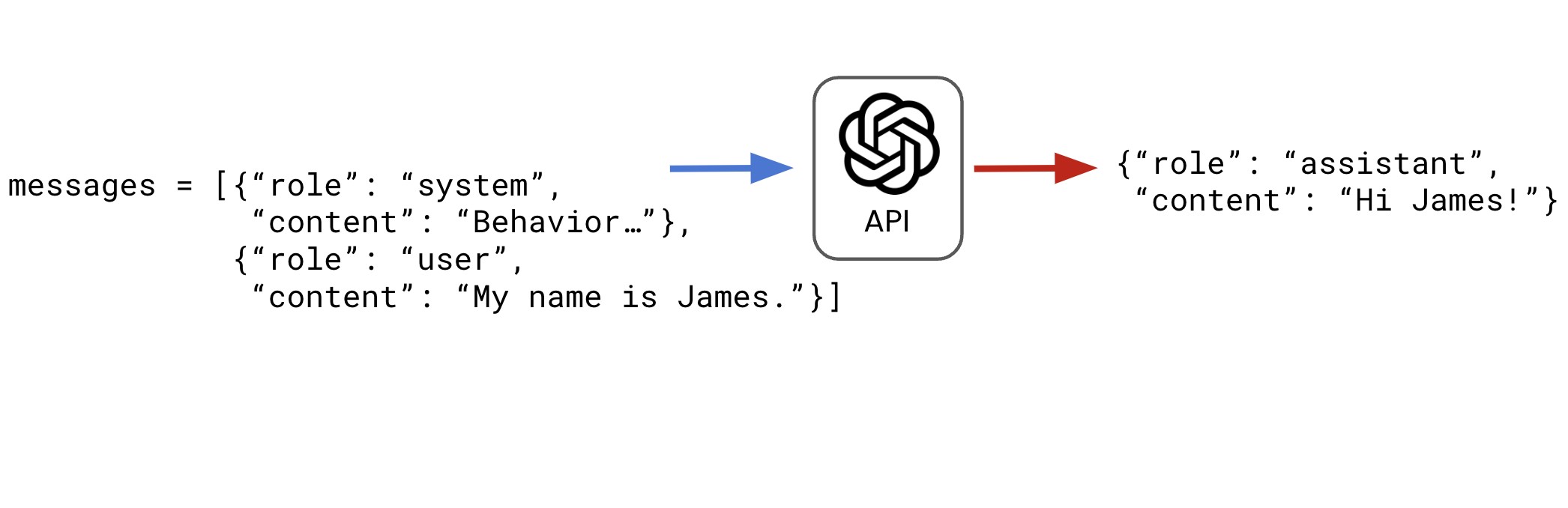
)

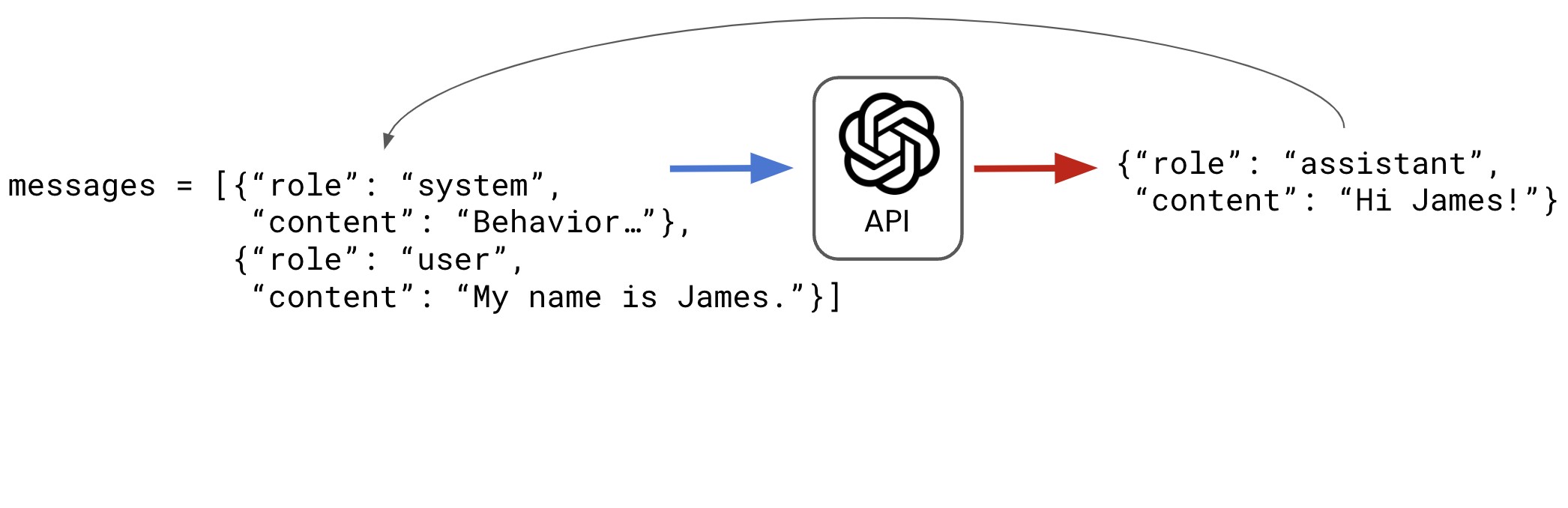
Mutable objects are objects whose values can change after they are created. Examples of mutable objects in Python include lists, sets and dictionaries. Immutable objects are objects whose values cannot change after they are created. Examples of immutable objects in Python include strings, numbers and tuples.

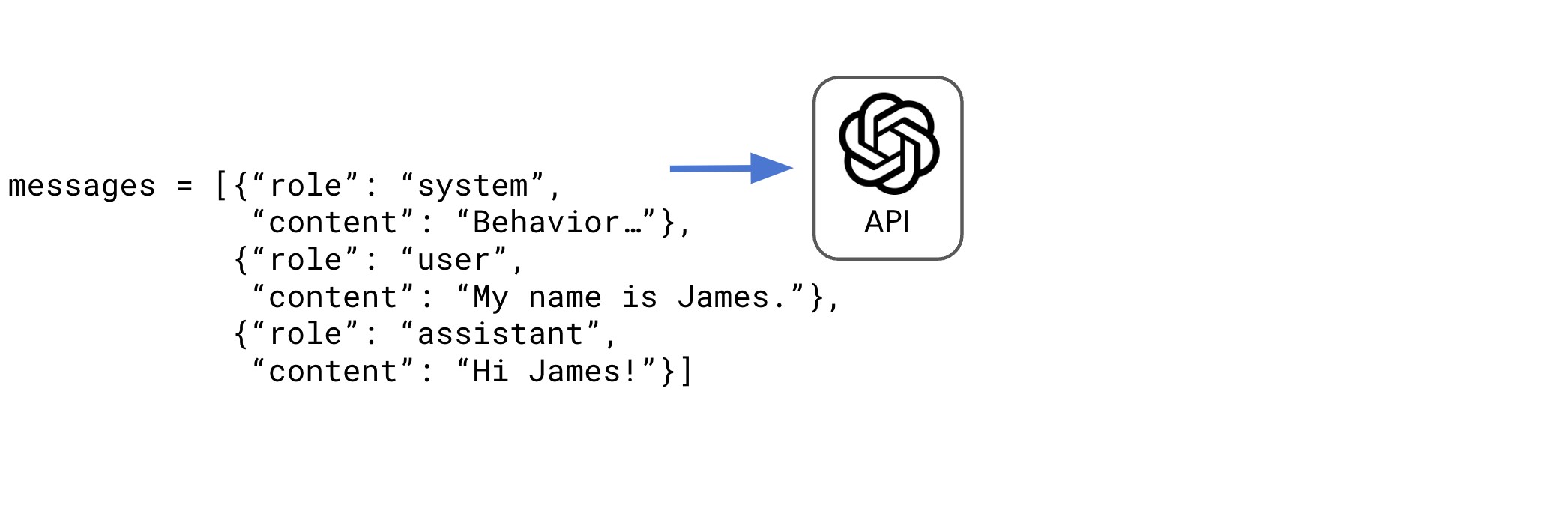
# Storing responses

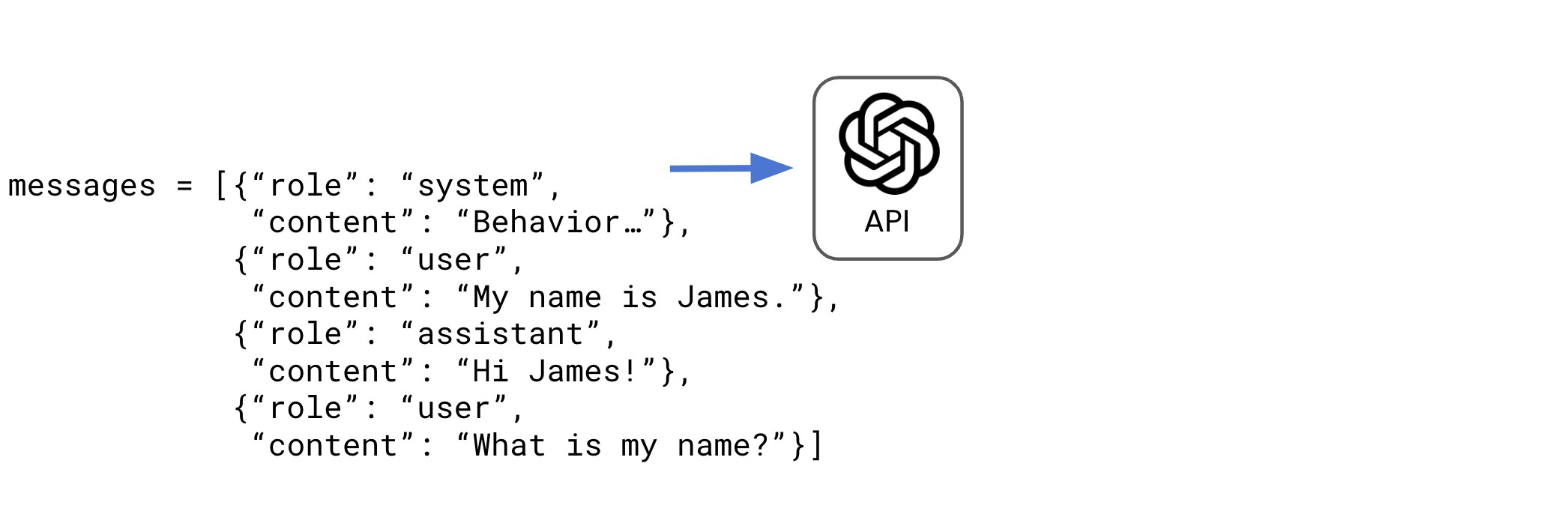
 Create conversation history

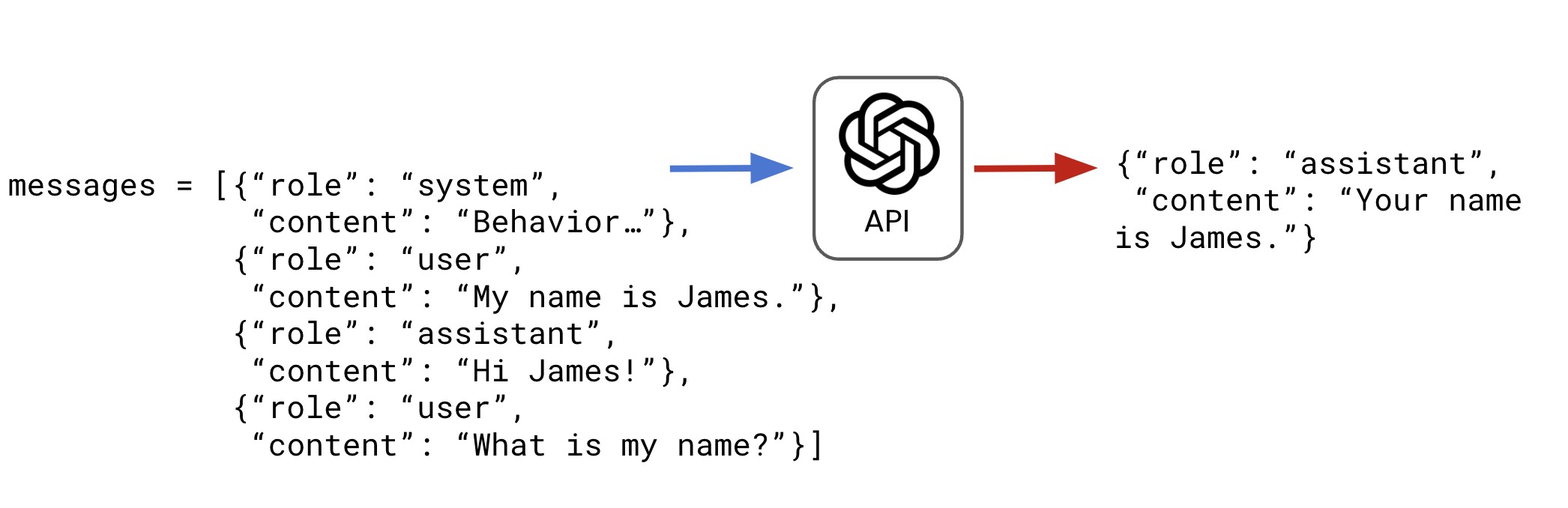
Create back-and-forth conversations











# Coding a conversation

y is Python so popular?

messages = [{

"role"

:

"system"

,

"content"

:

"You are a data science tutor who provides short, simple explanations."

}]

user\_qs = [

"Why is Python so popular?"

,

"Summarize this in one sentence."

]

for

q

~~i~~

n

user\_qs:

print

(

"User: "

, q)

user\_dict = {

"role"

:

"user"

,

"content"

:

q}

messages.append(user\_dict)

response = client.chat.completions.create(

model=

"gpt-4o-mini"

,

messages=messages

)

assistant\_dict = {

"role"

:

"assistant"

,

"content"

:

response.choices

[

0

]

.message.content

}

messages.append(assistant\_dict)

print

(

"Assistant: "

, response.choices[

0

]

.message.content,

"\n"

)

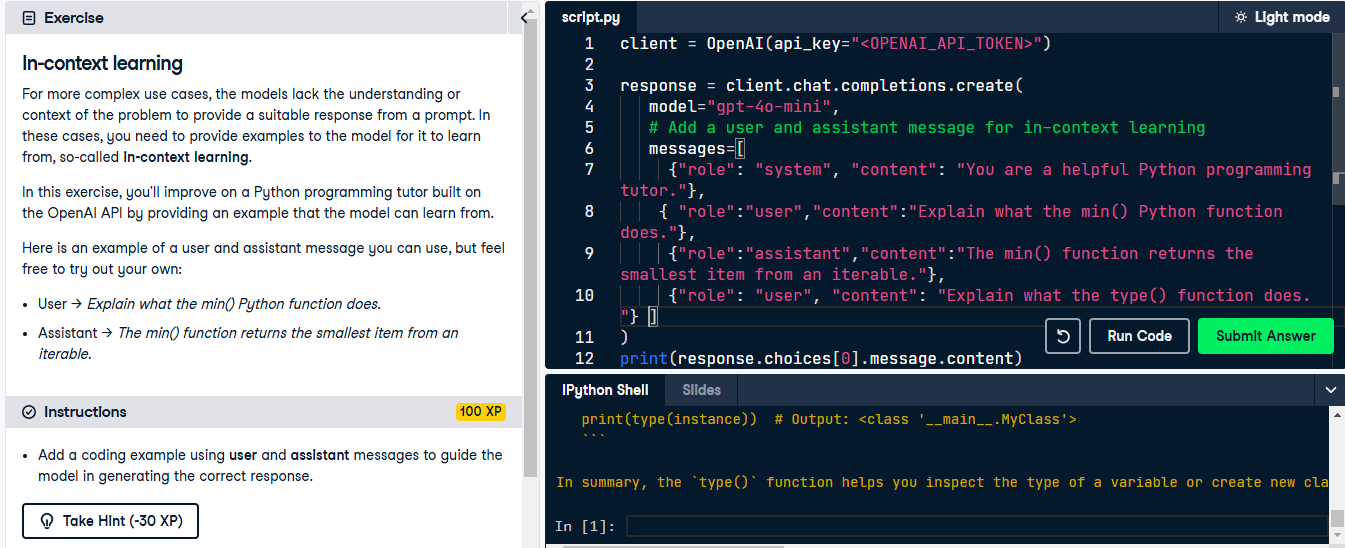
Assistant: Python is popular for many reasons, including its simplicity, versatility, and wide range of available libraries. It has a relatively easy-to-learn syntax that makes it accessible to beginners and experts alike. It can be used for a variety of tasks, such as data analysis, web development, scientific computing, and machine learning. Additionally, Python has an active community of developers who contribute to its development and share their knowledge through online resources and forums.

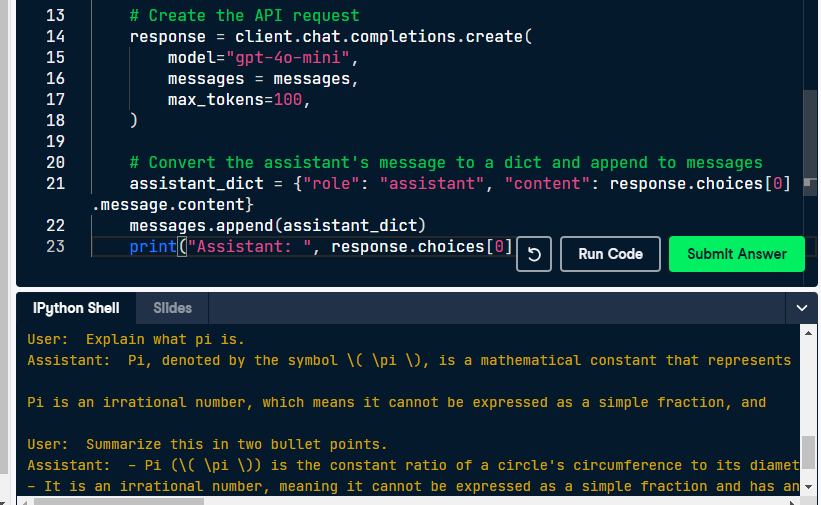
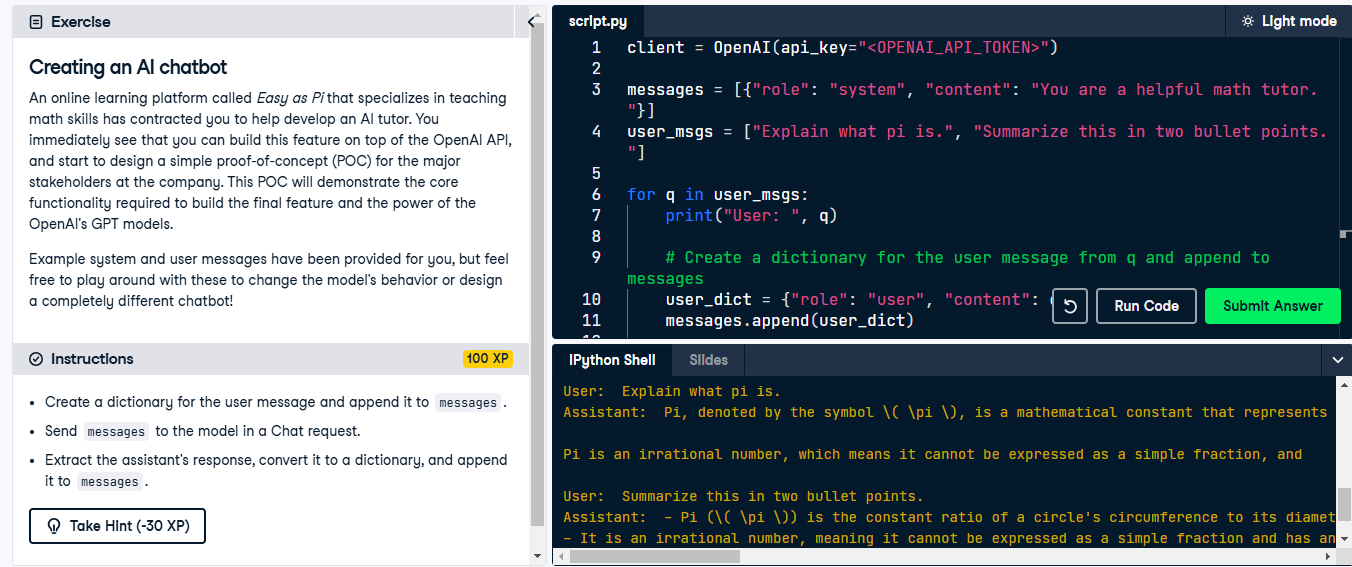
User: Summarize this in one sentence.

Assistant: Python is popular due to its simplicity, versatility, wide range of libraries, and active community of developers.

**Let's practice!**

**WORKING WITH THE OPENAI API**





1. https://openai.com/pricing [↑](#footnote-ref-1)