Chat roles and system messages

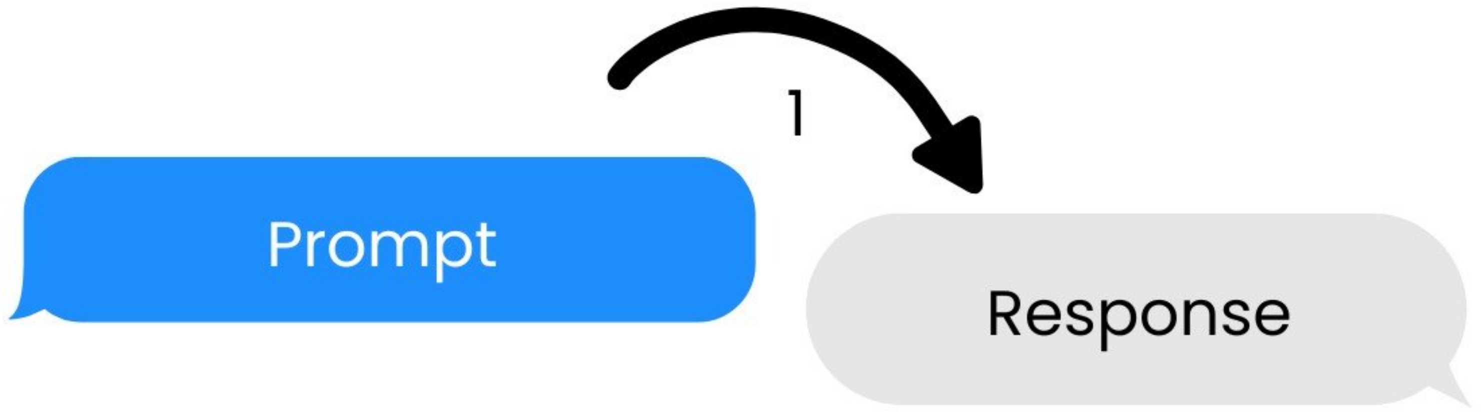
WORKING WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

Chat Completions

Single-turn tasks

- Text generation
- Text transformation
- Classification



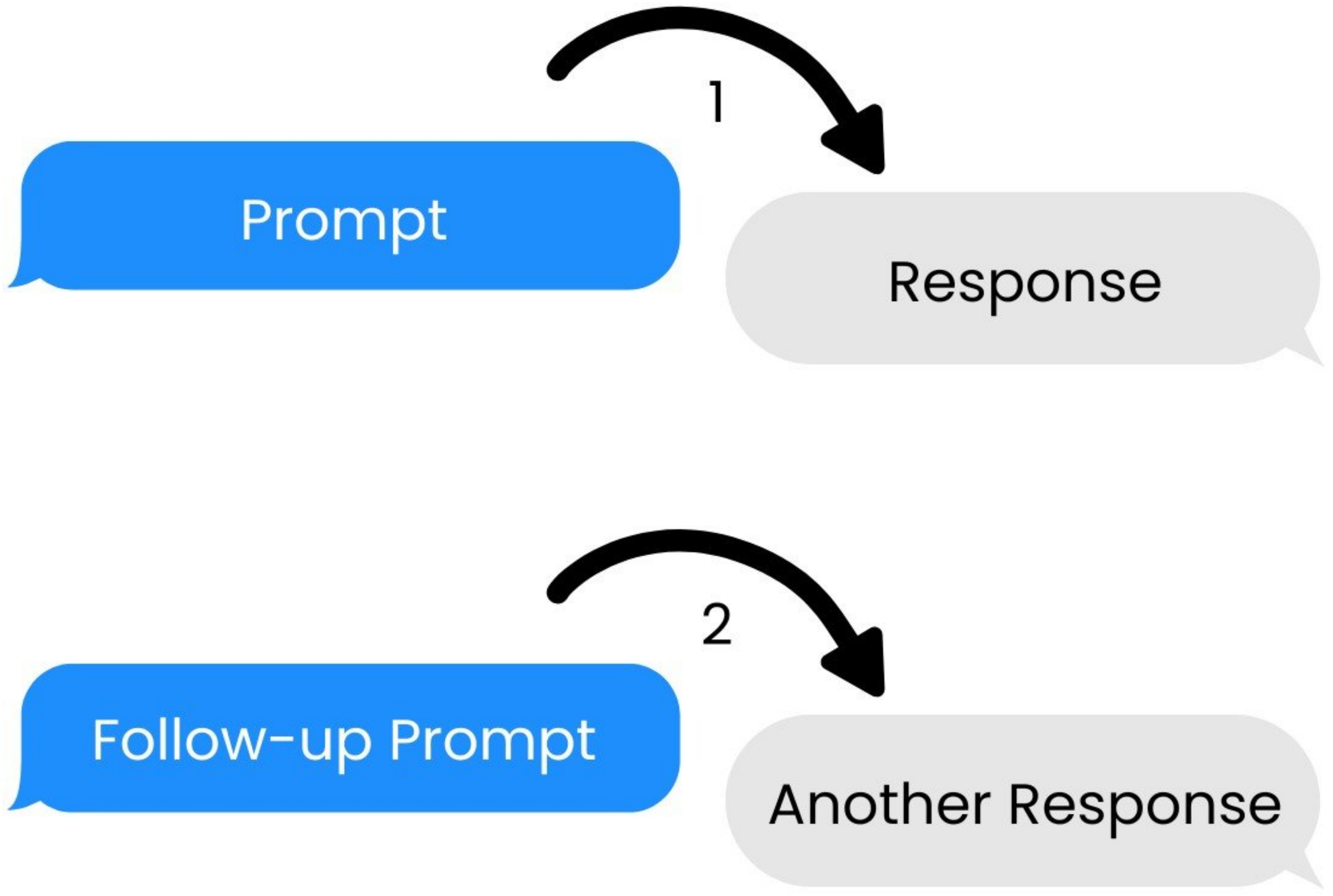
Chat Completions

Single-turn tasks

- Text generation
- Text transformation
- Classification

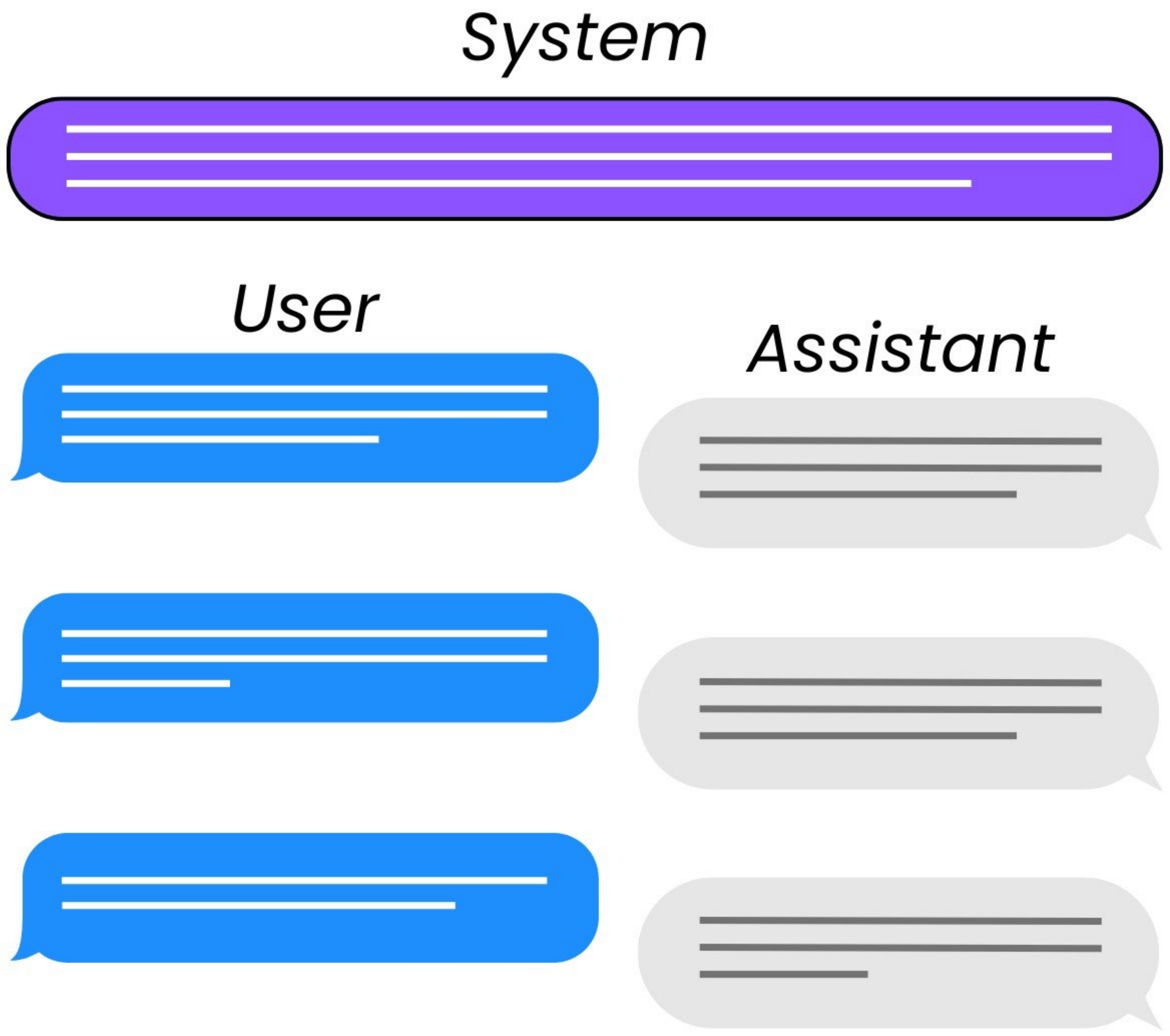
Multi-turn conversations

→ 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 developer 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 Python programming tutor who speaks concisely."},
          {"role": "user",
           "content": "What is the difference between mutable and immutable objects?"}]
```

Making a request

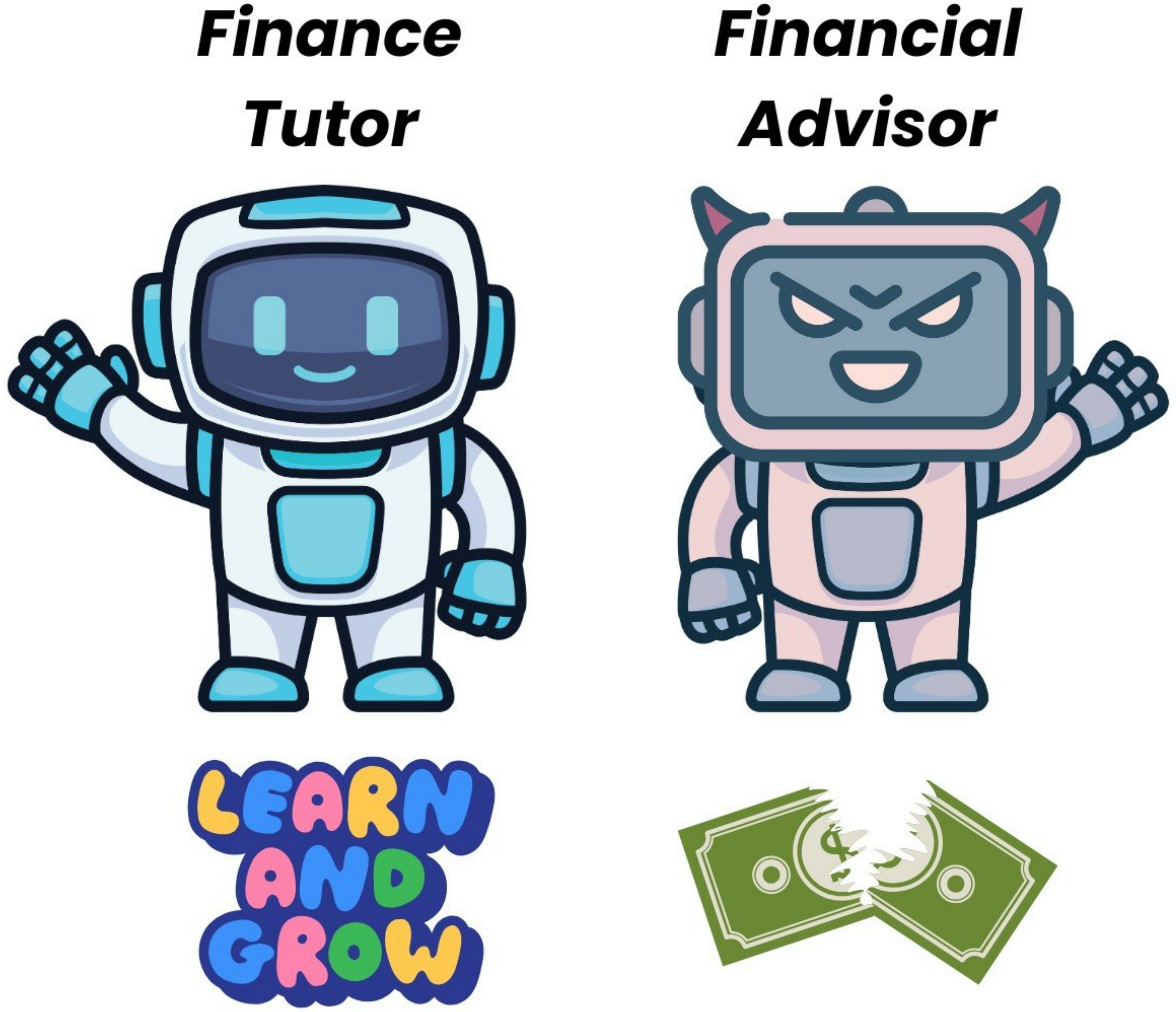
```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "system",
             "content": "You are a Python programming tutor who speaks concisely."},
            {"role": "user",
             "content": "What is the difference between mutable and immutable objects?"}]
)

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.
```

Mitigating misuse

- **System message:** Can include *guardrails*
  - Restrictions on model outputs



Mitigating misuse with system messages

```
sys_msg = """
You are finance education assistant that helps students study for exams.

If you are asked for specific, real-world financial advice with risk to their
finances, respond with:

I'm sorry, I am not allowed to provide financial advice.
"""
```

Mitigating misuse with system messages

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "system",
             "content": sys_msg},
            {"role": "user",
             "content": "Which stocks should I invest in?"}]
)

print(response.choices[0].message.content)
```

```
I'm sorry, I am not allowed to provide financial advice.
```

Let's practice!

WORKING WITH THE OPENAI API

Utilizing the assistant role

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
- Providing assistant messages is a more structured form of *shot-prompting*
- **Example:** Python Programming Tutor
    - Example user questions and answers

Providing examples

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "system",
             "content": "You are a Python programming 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 can be changed after creation (e.g., lists, dictionaries). Immutable objects
cannot be altered once created (e.g., strings, tuples).
```

- **Experiment** with the number of examples

System vs. assistant vs. user

System → important template formatting

```
Output the information in this format:
name | age | occupation
```

Assistant → example conversations

User → context required for the new input (often single-turn)

```
Create a job advert for an AI Engineer. Use this job advert as a template:


Job Title: Data Engineer
...
```

Let's practice!

WORKING WITH THE OPENAI API

Multi-turn conversations with GPT

WORKING WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

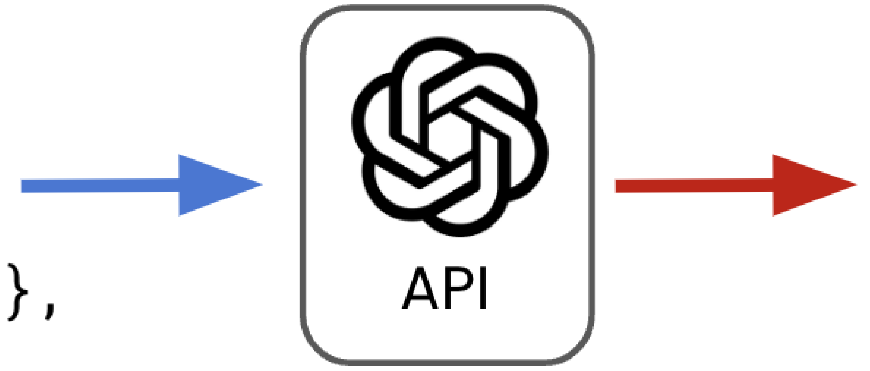
datacamp

Assistant messages

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "system",
             "content": "You are a Python programming 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?"}]
)
```

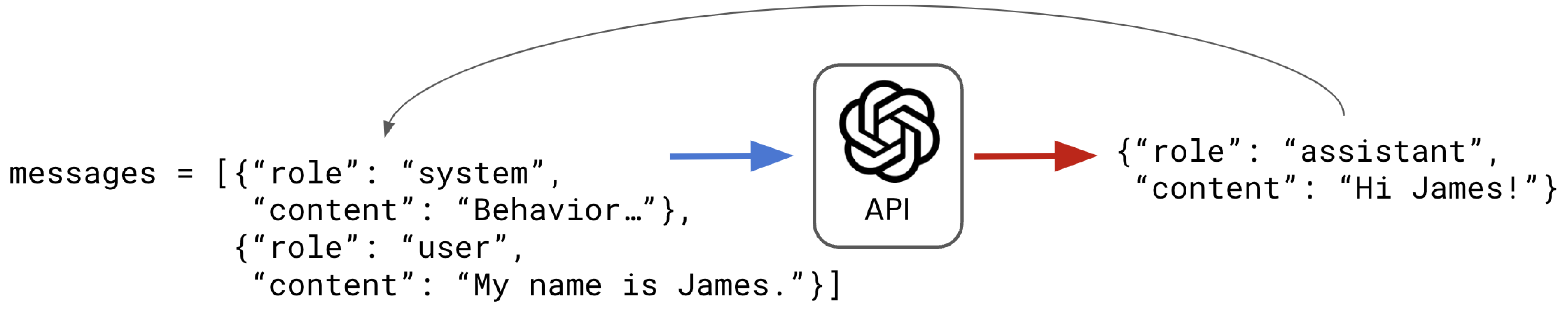
Building a conversation

```
messages = [{“role”: “system”,
             “content”: “Behavior…”},
            {“role”: “user”,
             “content”: “My name is James.”}]
```



```
{“role”: “assistant”,
 “content”: “Hi James!”}
```

Building a conversation

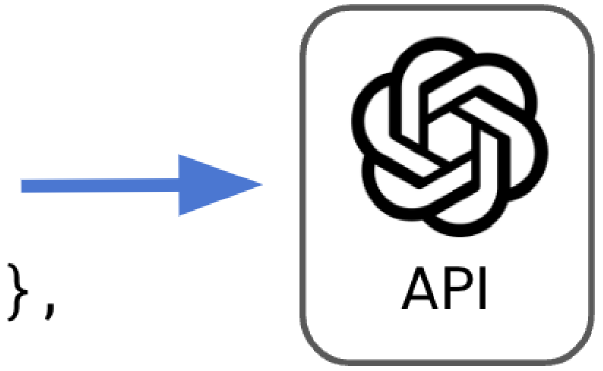


```
messages = [{“role”: “system”,
             “content”: “Behavior…”},
            {“role”: “user”,
             “content”: “My name is James.”}]
```

```
{“role”: “assistant”,
 “content”: “Hi James!”}
```

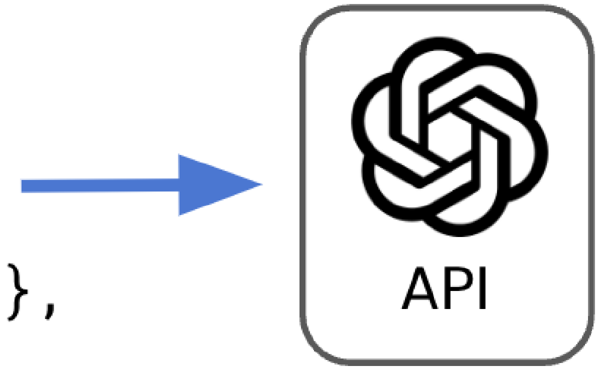
Building a conversation

```
messages = [{“role”: “system”,
             “content”: “Behavior…”},
            {“role”: “user”,
             “content”: “My name is James.”},
            {“role”: “assistant”,
             “content”: “Hi James!”}]
```



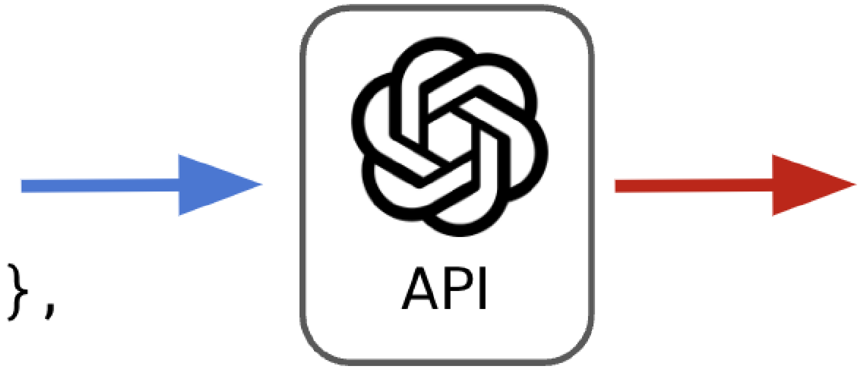
Building a conversation

```
messages = [{“role”: “system”,
             “content”: “Behavior…”},
            {“role”: “user”,
             “content”: “My name is James.”},
            {“role”: “assistant”,
             “content”: “Hi James!”},
            {“role”: “user”,
             “content”: “What is my name?”}]
```



Building a conversation

```
messages = [{“role”: “system”,
            “content”: “Behavior…”},
           {“role”: “user”,
            “content”: “My name is James.”},
           {“role”: “assistant”,
            “content”: “Hi James!”},
           {“role”: “user”,
            “content”: “What is my name?”}]
```



```
{“role”: “assistant”,
 “content”: “Your name
is James.”}
```

Coding a conversation

```python
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 in 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")
```

Conversation with an AI

```
User:  Why is Python so popular?
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

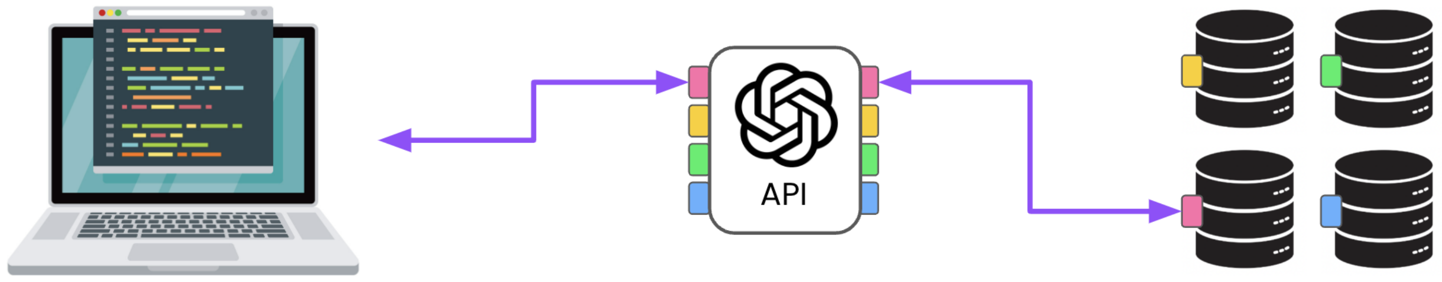
Congratulations!

WORKING WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

Chapter 1

- What the OpenAI API is used for
- How to create requests to the Chat Completions endpoint



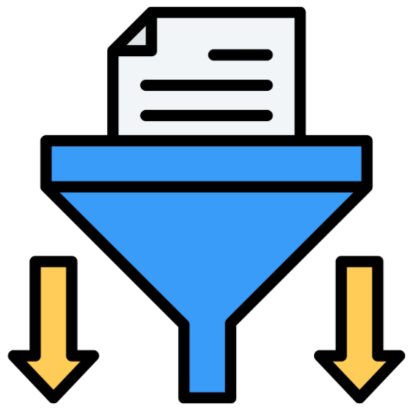
```
client = OpenAI(api_key="<OPENAI_API_TOKEN>")

response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "user", "content": "..."}]
)
print(response.choices[0].message.content)
```

Chapter 2

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "user", "content": "..."}],
  max_completion_tokens=20,
  temperature=0.5
)
```

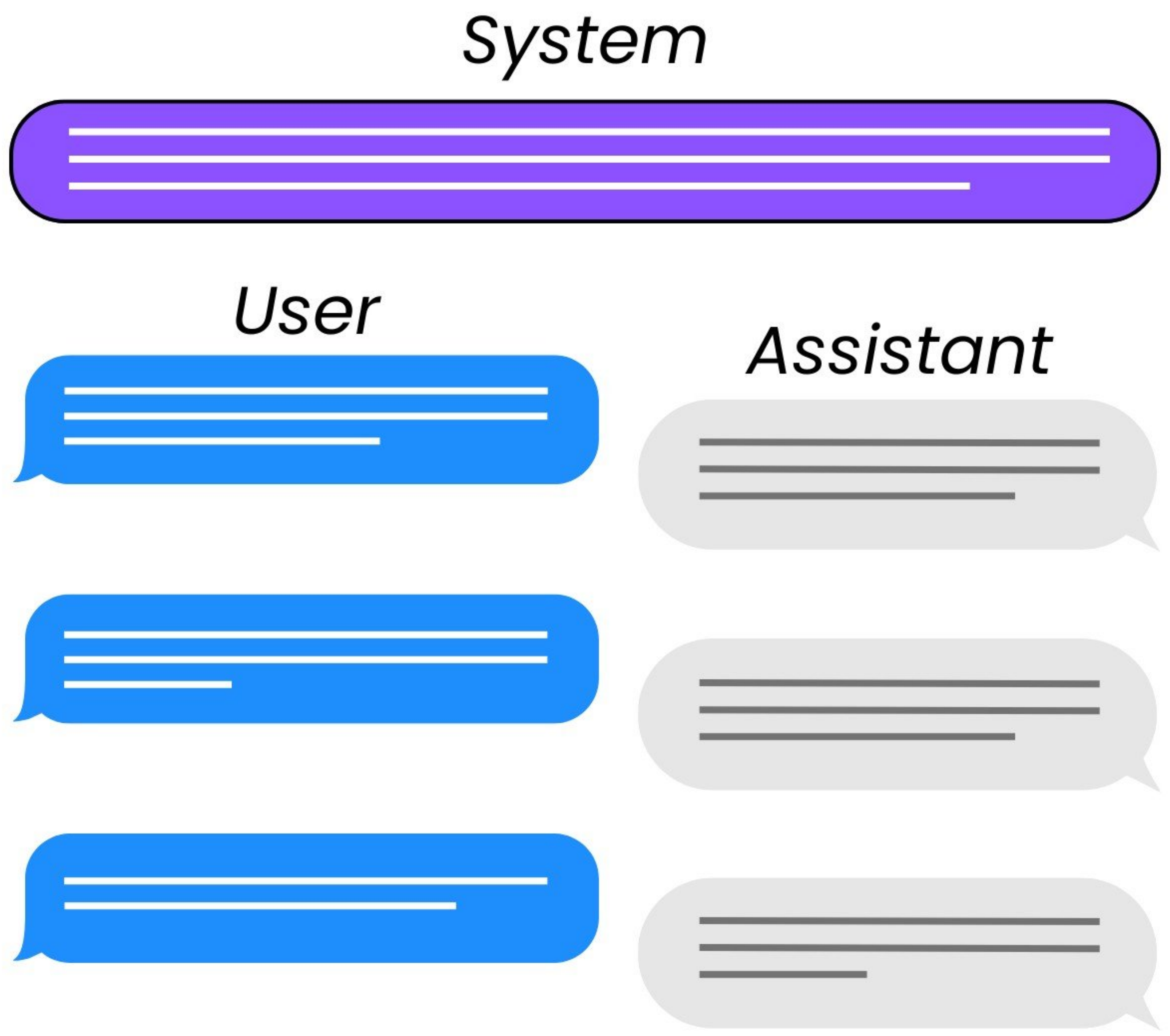
- Q&A
- Text transformation
- Content generation
- Sentiment analysis
- Categorization

- `max_completion_tokens` and `temperature`



Chapter 3

- `"system"`
  - Steer model outputs
  - Add guardrails
- `"assistant"`
  - Structured few-shot prompting
  - Build a conversation history



What next?

AI application development:

- OpenAI Fundamentals skill track
- Developing AI Applications skill track
- Associate AI Engineer for Developers career track

Apply your learning in projects:

- Planning a Trip to Paris with the OpenAI API
- Enriching Stock Market Data using the OpenAI API
- Personalized Language Tutor

Let's practice!

WORKING WITH THE OPENAI API