Text completions

WORKING WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

datacamp

Recap...

- Text completions for **Q&A**

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="How many days are in October?"
)

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

```
October has 31 days.
```

What is a text completion?

- Text *most likely* to complete the prompt

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="Life is like a box of chocolates."
)

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

```
You never know what you're going to get.
```

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

[1] Quote from Forrest Gump (1994)

Controlling response randomness

- `temperature` : control on *determinism*
- Ranges from `0` (highly deterministic) to `2` (very random)

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="Life is like a box of chocolates.",
  temperature=2
)

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

```
You never know what bitter constraints to any fate journey of Enlightenment you
instead be forced...
```

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:

Joanne is a Content Developer at DataCamp. Her favorite programming language is R,
which she uses for her statistical analyses.
"""
```

Content transformation

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt=prompt
)

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

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

Content generation

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="Create a tagline for a new hot dog stand."
)

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

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

Controlling response length

Default max_tokens

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="Write a haiku about AI."
)

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

```
AI so powerful
Computers that think and learn
Superseding
```

max_tokens = 30

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  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

In English:

- 1 token ~ 4 characters
- 100 tokens ~ 75 words

Example: 150 words → `max_tokens=200`

Returning to cost

- Increasing `max_tokens` increases cost
- Usage costs dependent on amount of generated text
  - Models are priced by *cost/1K tokens*
  - Input and output tokens can be priced differently
- Scoping feature cost often starts with a rough calculation:

$$\frac{\text{Cost}}{\text{Time}} = \text{Avg. Tokens Generated} \times \text{Model Cost} \times 1000 \times \frac{\text{Expected no. of requests}}{\text{Time}}$$

[1] https://openai.com/pricing

Let's practice!

WORKING WITH THE OPENAI API

Text completions for classification tasks

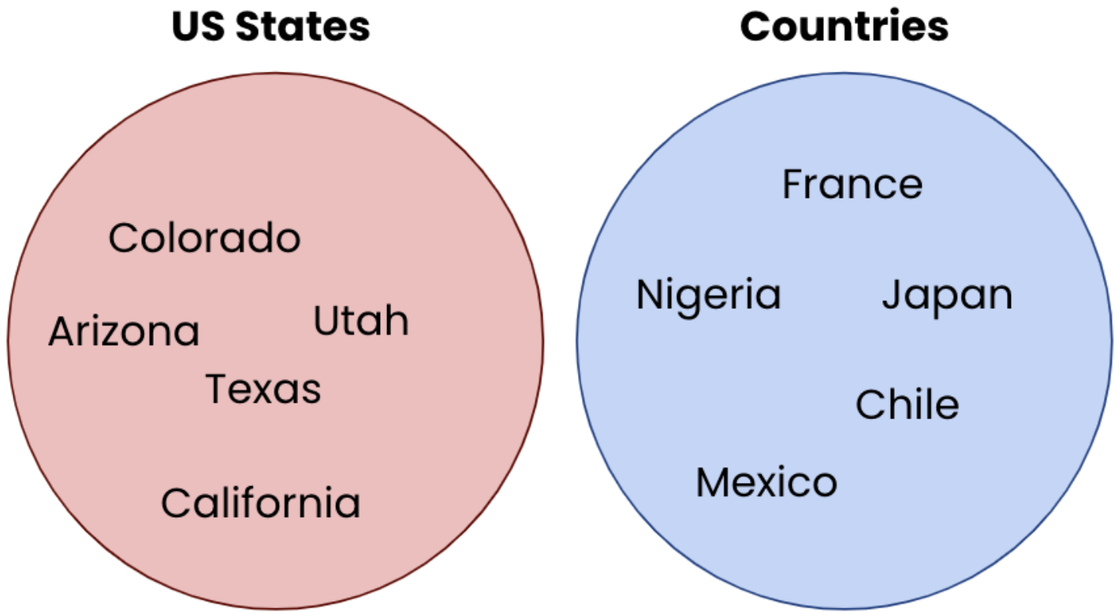
WORKING WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

datacamp

Classification tasks

- Task that involves assigning a label to information
  - Identifying the language from text
  - Categorization
  - Classifying sentiment



- Completions endpoint can perform these tasks, providing:
  - Model has sufficient knowledge
  - Prompt contains sufficient context

Categorizing animals

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="Classify the following animals into categories: zebra, crocodile,
          blue whale, polar bear, salmon, dog.",
  max_tokens=50
)

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

```
Mammals: Zebra, Polar Bear, Dog
Fish: Salmon
Reptiles: Crocodile
```

Specifying groups

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="Classify the following animals into animals with fur and without: zebra,
  crocodile, dolphin, polar bear, salmon, dog.",
  max_tokens=50
)

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

```
Animals with fur: Dog, Polar Bear, Zebra
Animals without fur: Crocodile, Dolphin, Salmon
```

Classifying sentiment

```
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.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt=prompt,
  max_tokens=50
)
print(response.choices[0].text)
```

Classifying sentiment

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
```

Classifying sentiment

```
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: *a handful of examples* provided

One-shot prompting

```
prompt = """Classify sentiment in the following statements:
The service was very slow // Disgruntled
Meal was decent, but I've had better. //
"""

response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt=prompt
)

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

```
Neutral
```

Few-shot prompting

```
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.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt=prompt
)

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

```
Mildly dissatisfied
```

Let's practice!

WORKING WITH THE OPENAI API

Chat completions with GPT

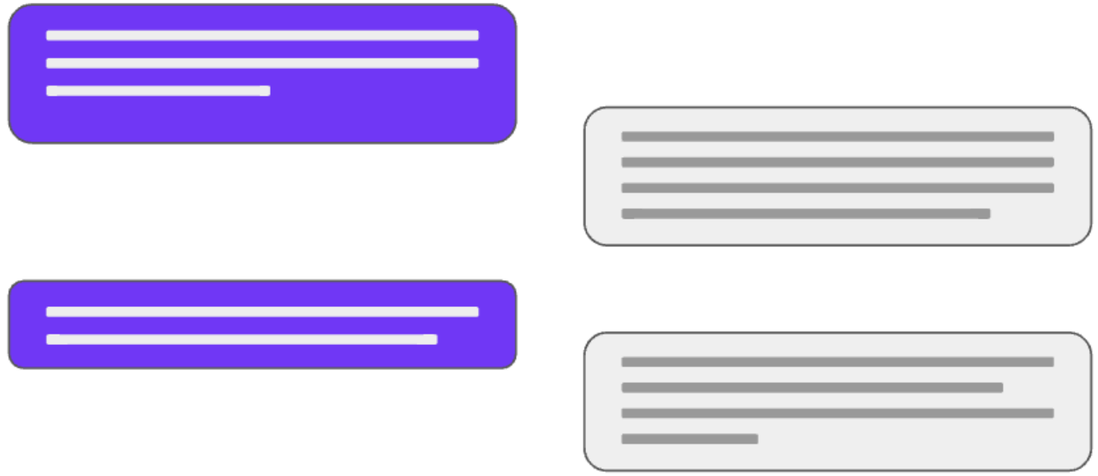
WORKING WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

datacamp

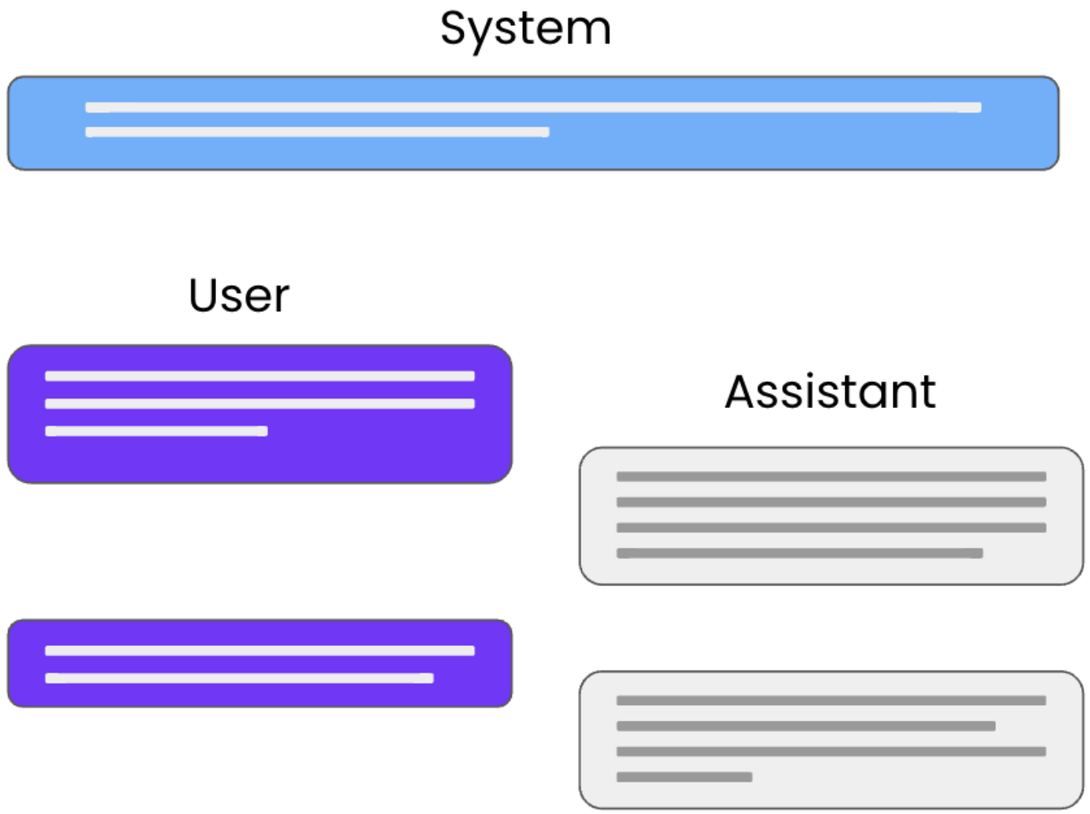
The Chat Completions endpoint

- Multi-turn conversations
  - Also performs well on single-turn
- Better customizability of response through the use of *roles*
- Cost benefit: `gpt-3.5-turbo` is cheaper than `gpt-3.5-turbo-instruct`



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

Completions

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="____"
)
```

Chat Completions

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=____
)
```

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-3.5-turbo",
  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)
```

The response

```
ChatCompletion(id='chatcmpl-8SM9TkxEkunyu6gLVJv3OvSmnxXvX',
               choices=[Choice(finish_reason='stop',
                               index=0,
                               message=ChatCompletionMessage(content='Mutable objects can be modified after
                               they are created, whereas immutable objects cannot be modified once they are
                               created.', role='assistant', function_call=None, tool_calls=None))],
               created=1701769027,
               model='gpt-3.5-turbo-0613',
               object='chat.completion',
               system_fingerprint=None,
               usage=CompletionUsage(completion_tokens=21, prompt_tokens=33, total_tokens=54))
```

Extracting the text

```
print(response.choices)
```

```
[Choice(finish_reason='stop',
        index=0,
        message=ChatCompletionMessage(content='Mutable objects can be modified after
                                they are created, whereas immutable objects cannot be modified once they are
                                created.', role='assistant', function_call=None, tool_calls=None))]
```

Extracting the text

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

```
Choice(finish_reason='stop',
       index=0,
       message=ChatCompletionMessage(content='Mutable objects can be modified after they
           are created, whereas immutable objects cannot be modified once they are created.',
           role='assistant', function_call=None, tool_calls=None))
```

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

Extracting the text

```
Mutable objects can be modified after they are created, whereas 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

datacamp

Chat completions for single-turn tasks

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  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-3.5-turbo",
  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?"}]
)
```

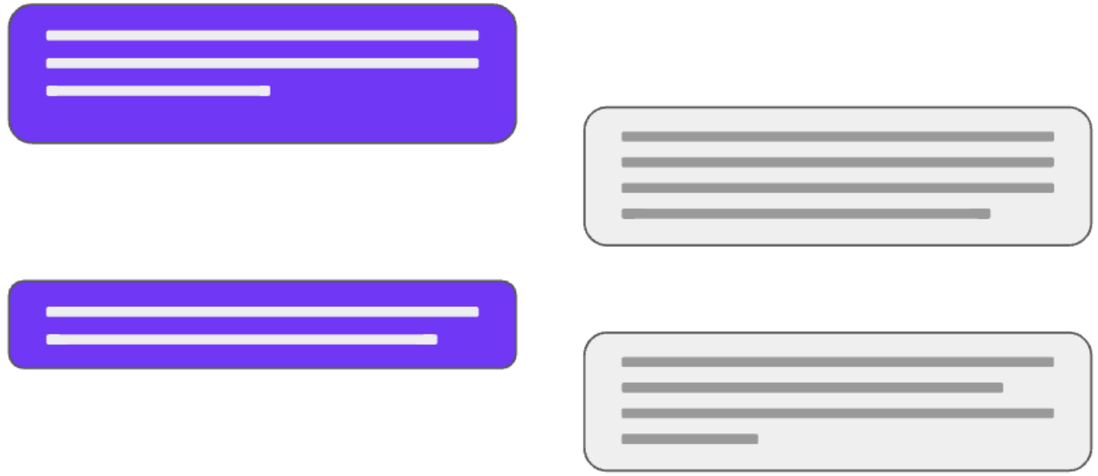
The response

```
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.
```

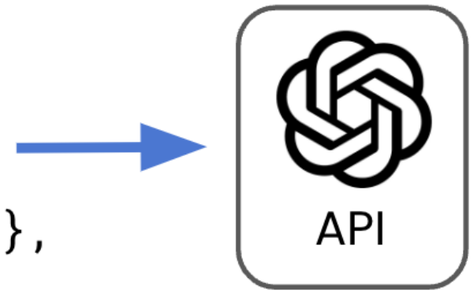
Storing responses

- Create conversation history
- Create back-and-forth conversations



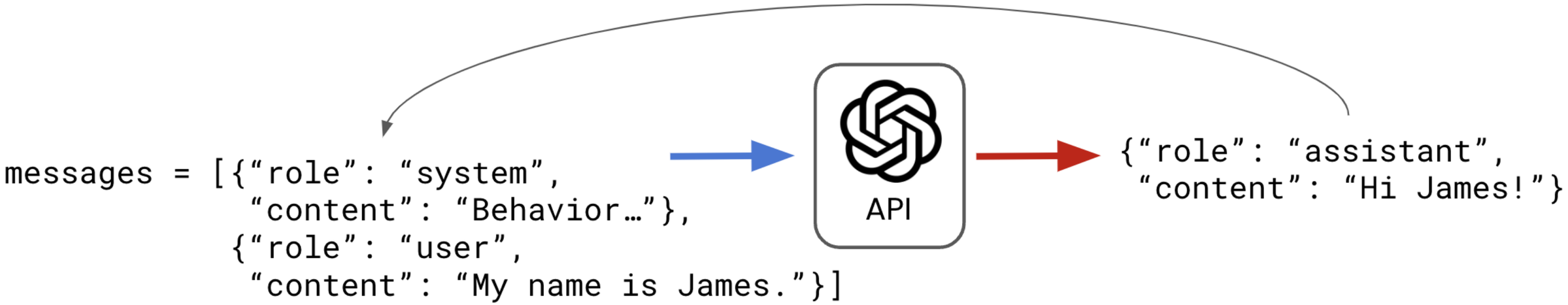
Building a conversation

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



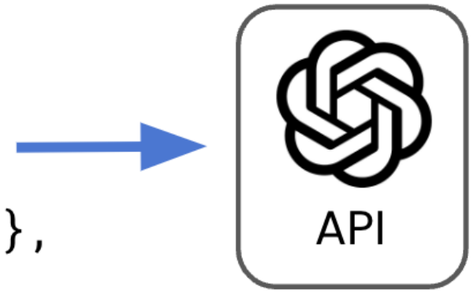

```
{“role”: “assistant”,
 “content”: “Hi James!”}
```

Building a conversation



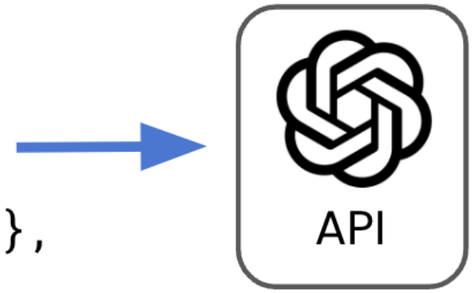
Building a conversation

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



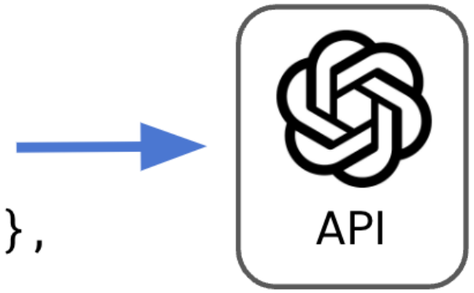
Building a conversation

```
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

```
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-3.5-turbo",
        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