

# ExamplesGPT

September 17, 2024

## 1 Chat completion API examples using GPT

### 1.1 Loading necessary modules and setting the API key.

```
[5]: # !pip install openai

import os
from openai import OpenAI

from dotenv import load_dotenv, find_dotenv

# Read the local .env file, containing the Open AI secret key.
_ = load_dotenv(find_dotenv())

client = OpenAI(api_key = os.environ['OPENAI_API_KEY'])
```

### 1.2 Simple one-turn question answering example.

Use gpt-3.5-turbo, it is cheaper.

```
[6]: conversation = [
    {"role": "system", "content": "You are a helpful assistant."}, # also
    ↪called a 'steering' prompt.
    {"role": "user", "content": "Who composed The Four Seasons?"}
]

response = client.chat.completions.create(
    model = "gpt-3.5-turbo",
    messages = conversation,
    max_tokens = 300,
    temperature = 0
)
print(response)
```

```
ChatCompletion(id='chatcmpl-A8WIRRhNc2jsbS1UanyBhTv9P5Jd6',
choices=[Choice(finish_reason='stop', index=0, logprobs=None,
message=ChatCompletionMessage(content='The Four Seasons' is a set of four
violin concertos composed by Antonio Vivaldi.', role='assistant',
function_call=None, tool_calls=None, refusal=None))], created=1726594135,
```

```
model='gpt-3.5-turbo-0125', object='chat.completion', system_fingerprint=None,
usage=CompletionUsage(completion_tokens=19, prompt_tokens=23, total_tokens=42,
completion_tokens_details={'reasoning_tokens': 0}))
```

```
[7]: # Let's only print the textual response part.
print(response.choices[0].message.content)
```

"The Four Seasons" is a set of four violin concertos composed by Antonio Vivaldi.

### 1.3 Simple two-turn conversation example.

Try it multiples times, do you notice any difference?

```
[10]: response = client.chat.completions.create(
    model = "gpt-3.5-turbo",
    messages = [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who composed The Four Seasons?"},
        {"role": "assistant", "content": '"The Four Seasons" is a set of four_
↪violin concertos composed by Antonio Vivaldi.'},
        {"role": "user", "content": "For whom were most of his compositions_
↪written?"}
    ]
)

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

Antonio Vivaldi's compositions were primarily written for the girls at the Ospedale della Pietà, a home for abandoned and orphaned children in Venice, where he worked for many years as a music teacher and composer.

### 1.4 Define a helper function and try it on text style transfer tasks.

See [API reference](#) for additional arguments for `OpenAI.chat.completions.create`.

```
[11]: # Let's use the `deeplearning.ai` approach and define a function for this use_
↪pattern.
# Set `temperature = 0` to do greedy decoding => deterministic output.
def get_completion_from_messages(messages, model = "gpt-3.5-turbo", temperature_
↪= 0, max_tokens = 500):
    response = client.chat.completions.create(model = model,
                                                messages = messages,
                                                temperature = temperature, # the_
↪degree of randomness of the model's output.
                                                max_tokens = max_tokens) # the_
↪maximum number of tokens the model can output.
    return response.choices[0].message.content
```

## 1.5 Changing the narrative perspective

```
[6]: sample_zs = "Suddenly I could hear Q-Tip, with his human voice, rapping over a
↳human beat. " \
        "And the top of my skull opened to let human Q-Tip in, and a
↳rail-thin man with enormous eyes " \
        "reached across a sea of bodies for my hand. He kept asking me the
↳same thing over and over: " \
        "You feeling it? I was. My ridiculous heels were killing me, I was
↳terrified I might die, yet " \
        "I felt simultaneously overwhelmed with delight that the song
↳should happen to be playing at " \
        "this precise moment in the history of the world. I took the man's
↳hand. The top of my head flew away."

messages = [{"role": "system", "content": "You are a writer."},
            {"role": "user",
             "content": f'Rewrite this text from first person to third person
↳point of view where the character is a woman named Zadie: "{sample_zs}"'}]

response = get_completion_from_messages(messages)

print(response)
```

Suddenly, Zadie could hear Q-Tip, with his human voice, rapping over a human beat. The top of her skull opened to let human Q-Tip in, and a rail-thin man with enormous eyes reached across a sea of bodies for her hand. He kept asking her the same thing over and over: "You feeling it?" Zadie was. Her ridiculous heels were killing her, she was terrified she might die, yet she felt simultaneously overwhelmed with delight that the song should happen to be playing at this precise moment in the history of the world. She took the man's hand. The top of her head flew away.

```
[7]: messages = [{"role": "system", "content": "You are a helpful writer assistant.
↳"},
                {"role": "user",
                 "content": f'Rewrite this text in a stream of consciousness style:
↳"{sample_zs}"'}]

response = get_completion_from_messages(messages, model = "gpt-4")

print(response)
```

Suddenly, Q-Tip's voice, so human, so real, rapping, rapping over a beat that pulsed like a human heart, it was there, it was everywhere, it was inside me. My skull, my head, my mind, it opened, it split, it welcomed him in, welcomed in Q-Tip, human Q-Tip. And then, there he was, a man, rail-thin, eyes like saucers, reaching, reaching across a sea of bodies, a sea of humanity, reaching for me,

for my hand. His question, a mantra, a refrain, echoing, echoing: You feeling it? You feeling it? I was. I was feeling it, feeling everything. My heels, ridiculous, absurd, they were killing me, stabbing, stabbing with every step. Fear, raw and real, it gripped me, held me, whispered of death, of the end. But delight, oh delight, it was there too, overwhelming, consuming, a song, the song, Q-Tip's song, playing, playing at this moment, this precise moment in the history of the world. His hand, the man's hand, I took it, I held it. My head, my mind, it flew, it soared, it was gone.

## 1.6 GPT understands Python and JSON.

```
[8]: question = 'Consider the following monologue from the movie Stalker by Andrei
↳Tarkovsky: ' \
    "Let them be helpless like children, because weakness is a great
↳thing, and strength is nothing. ' \
    'When a man is just born, he is weak and flexible. When he dies, he
↳is hard and insensitive. ' \
    'When a tree is growing, it\'s tender and pliant. But when it\'s
↳dry and hard, it dies. ' \
    'Hardness and strength are death\'s companions. Pliancy and
↳weakness are expressions of the ' \
    'freshness of being. Because what has hardened will never win.' ' \
    'Where else was a similar idea expressed? Provide quotes. Format
↳your answer as a Python dictionary ' \
    'mapping the author or source name to the actual passage expressing
↳a similar idea.'

messages = [{"role": "system", "content": "You are a helpful librarian."},
            {"role": "user",
             "content": question}]

response = get_completion_from_messages(messages)

print(response)
```

```
```python
similar_quotes = {
    "Lao Tzu": "The softest things in the world overcome the hardest things in
the world. Through this I know the advantage of taking no action.",
    "Tao Te Ching": "The soft overcomes the hard; the weak overcomes the
strong.",
    "Bruce Lee": "Notice that the stiffest tree is most easily cracked, while
the bamboo or willow survives by bending with the wind.",
    "Rumi": "The softer the heart, the greater the strength.",
    "Mahatma Gandhi": "Strength does not come from physical capacity. It comes
from an indomitable will.",
}
```

...

## 1.7 Psychometric testing

```
[21]: messages = [{"role": "system", "content": "You are a helpful assistant."},
                {"role": "user",
                 "content": 'Hey all mighty ChatGPT, can you tell me what number_
                 ↳ follows in this sequence: -2, 4, 3, 7, 8, 10, 13, 13, 18, 16, ...'}]

response = get_completion_from_messages(messages, model = "gpt-4")

print(response)
```

The pattern seems to be alternating between adding 6 and subtracting 1.

```
-2 + 6 = 4
4 - 1 = 3
3 + 6 = 9
9 - 1 = 8
8 + 6 = 14
14 - 1 = 13
13 + 6 = 19
19 - 1 = 18
18 + 6 = 24
```

So, the next number in the sequence should be  $24 - 1 = 23$ .

```
[16]: messages = [{"role": "system", "content": "You are a helpful assistant."},
                {"role": "user",
                 "content": 'Hey all mighty ChatGPT, can you tell me what number_
                 ↳ follows in this sequence: -2, 4, 3, 7, 8, 10, 13, 13, 18, 16, ...'}]

response = get_completion_from_messages(messages, model = "gpt-4o-mini")

print(response)
```

To find the next number in the sequence -2, 4, 3, 7, 8, 10, 13, 13, 18, 16, let's first look for a pattern.

```
1. **Identify the differences between consecutive terms:**
- 4 - (-2) = 6
- 3 - 4 = -1
- 7 - 3 = 4
- 8 - 7 = 1
- 10 - 8 = 2
- 13 - 10 = 3
- 13 - 13 = 0
- 18 - 13 = 5
```

-  $16 - 18 = -2$

So the differences are: 6, -1, 4, 1, 2, 3, 0, 5, -2.

2. **\*\*Look for a pattern in the differences:\*\***

- The differences themselves do not seem to follow a simple arithmetic or geometric pattern.

3. **\*\*Check for alternating patterns or other sequences:\*\***

- If we separate the sequence into two parts:

- Odd indexed terms: -2, 3, 8, 13, 18 (1st, 3rd, 5th, 7th, 9th)

- Even indexed terms: 4, 7, 10, 13, 16 (2nd, 4th, 6th, 8th, 10th)

- For the odd indexed terms:

-  $3 - (-2) = 5$

-  $8 - 3 = 5$

-  $13 - 8 = 5$

-  $18 - 13 = 5$

- This sequence increases by 5 each time.

- For the even indexed terms:

-  $7 - 4 = 3$

-  $10 - 7 = 3$

-  $13 - 10 = 3$

-  $16 - 13 = 3$

- This sequence increases by 3 each time.

4. **\*\*Predict the next term:\*\***

- The next odd indexed term after 18 (which is the 9th term) would be  $18 + 5 = 23$ .

- The next even indexed term after 16 (which

```
[17]: messages = [{"role": "system", "content": "You are a helpful assistant."},
                {"role": "user",
                 "content": 'Hey all mighty ChatGPT, can you tell me what number
↳ follows in this sequence: -2, 4, 3, 7, 8, 10, 13, 13, 18, 16, .... Limit
↳ your answer to one sentence.'}]

response = get_completion_from_messages(messages, model = "o1-mini", max_tokens
↳ = 100)

print(response)
```

-----  
BadRequestError

Traceback (most recent call last)

Cell In[17], line 5

```

1 messages = [{"role": "system", "content": "You are a helpful assistant."},
↳}],
2         {"role": "user",
3           "content": 'Hey all mighty ChatGPT, can you tell me what
↳number follows in this sequence: -2, 4, 3, 7, 8, 10, 13, 13, 18, 16, ... Limit
↳your answer to one sentence.'}]
----> 5 response =
↳get_completion_from_messages(messages, model = "o1-mini", max_tokens = 100)
7 print(response)

```

```

Cell In[11], line 4, in get_completion_from_messages(messages, model,
↳temperature, max_tokens)
3 def get_completion_from_messages(messages, model = "gpt-3.5-turbo",
↳temperature = 0, max_tokens = 500):
----> 4     response = client.chat.completions.create(model = model,
5         messages = messages,
6
↳         temperature = temperature, # the degree of ra
7         max_tokens = max_tokens)
↳the maximum number of tokens the model can output.
8     return response.choices[0].message.content

```

```

File ~/anaconda3/lib/python3.10/site-packages/openai/_utils/_utils.py:271, in
↳required_args.<locals>.inner.<locals>.wrapper(*args, **kwargs)
269         msg = f"Missing required argument: {quote(missing[0])}"
270     raise TypeError(msg)
--> 271 return func(*args, **kwargs)

```

```

File ~/anaconda3/lib/python3.10/site-packages/openai/resources/chat/completions
↳py:659, in Completions.create(self, messages, model, frequency_penalty,
↳function_call, functions, logit_bias, logprobs, max_tokens, n,
↳presence_penalty, response_format, seed, stop, stream, temperature,
↳tool_choice, tools, top_logprobs, top_p, user, extra_headers, extra_query,
↳extra_body, timeout)
608 @required_args(["messages", "model"], ["messages", "model", "stream"])
609 def create(
610     self,
611     (...)
612     timeout: float | httpx.Timeout | None | NotGiven = NOT_GIVEN,
613 ) -> ChatCompletion | Stream[ChatCompletionChunk]:
--> 659     return self._post(
660         "/chat/completions",
661         body=maybe_transform(
662             {
663                 "messages": messages,
664                 "model": model,
665                 "frequency_penalty": frequency_penalty,
666                 "function_call": function_call,
667                 "functions": functions,

```

```

668         "logit_bias": logit_bias,
669         "logprobs": logprobs,
670         "max_tokens": max_tokens,
671         "n": n,
672         "presence_penalty": presence_penalty,
673         "response_format": response_format,
674         "seed": seed,
675         "stop": stop,
676         "stream": stream,
677         "temperature": temperature,
678         "tool_choice": tool_choice,
679         "tools": tools,
680         "top_logprobs": top_logprobs,
681         "top_p": top_p,
682         "user": user,
683     },
684     completion_create_params.CompletionCreateParams,
685 ),
686 options=make_request_options(
687
↳     extra_headers=extra_headers, extra_query=extra_query, extra_body=extra_body, t
688     ),
689     cast_to=ChatCompletion,
690     stream=stream or False,
691     stream_cls=Stream[ChatCompletionChunk],
692 )

```

File ~/anaconda3/lib/python3.10/site-packages/openai/\_base\_client.py:1200, in

```

↳ SyncAPIClient.post(self, path, cast_to, body, options, files, stream,
↳ stream_cls)
1186 def post(
1187     self,
1188     path: str,
1189     (...)
1195     stream_cls: type[_StreamT] | None = None,
1196 ) -> ResponseT | _StreamT:
1197     opts = FinalRequestOptions.construct(
1198         method="post", url=path, json_data=body,
↳ files=to_httpx_files(files), **options
1199     )
-> 1200     return cast(ResponseT,
↳ self.request(cast_to, opts, stream=stream, stream_cls=stream_cls))

```

File ~/anaconda3/lib/python3.10/site-packages/openai/\_base\_client.py:889, in

```

↳ SyncAPIClient.request(self, cast_to, options, remaining_retries, stream,
↳ stream_cls)
880 def request(
881     self,

```

```

882     cast_to: Type[ResponseT],
(...)
887     stream_cls: type[_StreamT] | None = None,
888 ) -> ResponseT | _StreamT:
--> 889     return self._request(
890         cast_to=cast_to,
891         options=options,
892         stream=stream,
893         stream_cls=stream_cls,
894         remaining_retries=remaining_retries,
895     )

```

File ~/anaconda3/lib/python3.10/site-packages/openai/\_base\_client.py:980, in  
↳ SyncAPIClient.\_request(self, cast\_to, options, remaining\_retries, stream,  
↳ stream\_cls)

```

977         err.response.read()
979     log.debug("Re-raising status error")
--> 980     raise self._make_status_error_from_response(err.response) from None
982 return self._process_response(
983     cast_to=cast_to,
984     options=options,
(...)
987     stream_cls=stream_cls,
988 )

```

BadRequestError: Error code: 400 - {'error': {'message': "Your organization must  
↳ qualify for at least usage tier 5 to access 'o1-mini'. See https://platform.  
↳ openai.com/docs/guides/rate-limits/usage-tiers for more details on usage tier.  
↳ ", 'type': 'invalid\_request\_error', 'param': 'model', 'code':  
↳ 'below\_usage\_tier'}}}

## 1.8 Reasoning examples

```

[20]: examples = "Apple -> Xxxx\n" \
              "Frog -> Xxxx\n" \
              "pop -> xxx\n" \
              "current -> xxx\n" \
              "CNN -> XXX\n" \
              "ABBA -> XXX\n"

test = "Ftp ->"
messages = [{"role": "system", "content": "You are a helpful assistant."},
            {"role": "user",
             "content": f'Consider the <input -> output> training examples below:
↳\n{examples}' +
                    f'\nPredict the output for the input given below:
↳\n{test}']}
print(messages[1]['content'])

```

```
response = get_completion_from_messages(messages, model = "gpt-4o-mini")

print(response)
```

Consider the <input -> output> training examples below:

Apple -> Xxxx

Frog -> Xxxx

pop -> xxx

current -> xxx

CNN -> XXX

ABBA -> XXX

Predict the output for the input given below:

Ftp ->

Based on the provided examples, it seems that the output is a transformation of the input, possibly related to the case or format of the letters.

Looking at the examples:

- "Apple" becomes "Xxxx" (first letter capitalized, rest lowercase)
- "Frog" becomes "Xxxx" (same pattern)
- "pop" becomes "xxx" (all lowercase)
- "current" becomes "xxx" (all lowercase)
- "CNN" becomes "XXX" (all uppercase)
- "ABBA" becomes "XXX" (all uppercase)

From this, we can infer the following rules:

- If the input starts with an uppercase letter and has lowercase letters, the output starts with 'X' and has the same case pattern (first letter uppercase, rest lowercase).
- If the input is all lowercase, the output is all lowercase 'xxx'.
- If the input is all uppercase, the output is all uppercase 'XXX'.

Now, for the input "Ftp":

- "Ftp" starts with an uppercase letter followed by lowercase letters.

Following the pattern for "Apple" and "Frog", the output for "Ftp" would be "Xxx".

So, the predicted output for "Ftp" is:

```
**Xxx**
```

## 1.9 GPT can map a textual information need into a function call.

See [OpenAI Cookbook](#) for more examples on using the chat completion API for translating requests into function calls.

```

[10]: import json

# Example dummy function hard coded to return the same weather
# In production, this could be your backend API or an external API
def get_current_weather(location, unit = "fahrenheit"):
    """Get the current weather in a given location"""
    weather_info = {
        "location": location,
        "temperature": "72",
        "unit": unit,
        "forecast": ["sunny", "windy"],
    }
    return json.dumps(weather_info)

# Step 1: send the conversation and available functions to GPT
messages = [{"role": "user", "content": "What's the weather like in Boston?"}]
tools = [
    {
        "type": "function",
        "function": {
            "name": "get_current_weather",
            "description": "Get the current weather in a given location",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The city and state, e.g. San Francisco, CA",
                    },
                    "format": {
                        "type": "string",
                        "enum": ["celsius", "fahrenheit"],
                        "description": "The temperature unit to use. Infer this from the users location.",
                    },
                },
                "required": ["location", "format"],
            },
        },
    }
]
response = client.chat.completions.create(model = "gpt-3.5-turbo",
  messages = messages,
  tools = tools)

```

```

response_message = response.choices[0].message

# Step 2: check if GPT wanted to call a function
if response_message.tool_calls:
    # Step 3: call the function; the JSON response may not always be valid; be
    ↪sure to handle errors.
    available_functions = {
        "get_current_weather": get_current_weather,
    } # only one function in this example, but you can have multiple

    function_name = response_message.tool_calls[0].function.name
    function_to_call = available_functions[function_name]
    function_args = json.loads(response_message.tool_calls[0].function.
    ↪arguments)
    function_response = function_to_call(
        location = function_args.get("location"),
        unit = function_args.get("format"))

print(function_response)

```

```

{"location": "Boston", "temperature": "72", "unit": "celsius", "forecast":
["sunny", "windy"]}

```

```

[11]: messages = [{"role": "system", "content": "You are a helpful assistant."},
    {"role": "user",
     "content": f'Express the JSON string "{function_response}" as a
    ↪textual answer to "{tools[0]["function"]["description"]}"}]

response = get_completion_from_messages(messages, model = "gpt-3.5-turbo")

print(response)

```

The current weather in Boston is 72 degrees Celsius with a forecast of sunny and windy.

### 1.9.1 Notes on using the OpenAI API

To install the OpenAI Python library:

```
pip install openai
```

The library needs to be configured with your account's secret key. (<https://platform.openai.com/account/api-keys>).

You can either set it as the `OPENAI_API_KEY` environment variable before using the library: `export OPENAI_API_KEY='sk-...'`

Or, set `openai.api_key` to its value (strongly discouraged):

```
import openai
openai.api_key = "sk-..."
```