ExamplesGPT

September 25, 2025

1 Using Open AI GPT models through the Responses API

The Responses API is a new way to interact with OpenAI models, designed to be simpler and more flexible than previous APIs. It makes it easy to build advanced AI applications that use multiple tools, handle multi-turn conversations, and work with different types of data (not just text).

Unlike older APIs — such as Chat Completion APIs, which were built mainly for text, or the Assistants API, which can require a lot of setup — the Responses API is built from the ground up for:

- Seamless multi-turn interactions (carry on a conversation across several steps in a single API call).
- Easy access to powerful hosted tools (like file search, web search, and code interpreter).
- Fine-grained control over the context you send to the model.

As AI models become more capable of complex, long-running reasoning, developers need an API that is both asynchronous and stateful. The Responses API is designed to meet these needs.

In this guide, you'll see some of the new features the Responses API offers, along with practical examples to help you get started.

1.1 Loading necessary modules and setting the API key.

[1]:
```
# !pip install openai
# !pip install python-dotenv

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.

We can use `gpt-5`, `gpt-5-mini`, or `gpt-5-nano`. For very simple questions, use `gpt-5-nano`, it is cheapest.

[122]:
```
response1 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "minimal"},
    input = [
        {
            "role": "developer",
            "content": "You are a helpful music historian."
        },
        {
            "role": "user",
            "content": "Who composed The Four Seasons?"
        }
    ]
)

print(response1)
```

```
Response(id='resp_68d2dca6db7081938f6d6aeba63cee63016072867af7cd80',
created_at=1758649510.0, error=None, incomplete_details=None, instructions=None,
metadata={}, model='gpt-5-nano-2025-08-07', object='response', output=[ResponseR
easoningItem(id='rs_68d2dca76c308193876192340d6b3e97016072867af7cd80',
summary=[], type='reasoning', content=None, encrypted_content=None,
status=None),
ResponseOutputMessage(id='msg_68d2dca7896481939959dbb5159000e4016072867af7cd80',
content=[ResponseOutputText(annotations=[], text='The Four Seasons was composed
by Antonio Vivaldi. It is a set of four violin concertos, each representing a
season: Spring, Summer, Autumn, and Winter. They were published in 1725 as part
of his opus 8, Il cimento dell'armonia e dell'inventione.', type='output_text',
logprobs=[])], role='assistant', status='completed', type='message')],
parallel_tool_calls=True, temperature=1.0, tool_choice='auto', tools=[],
top_p=1.0, background=False, conversation=None, max_output_tokens=None,
max_tool_calls=None, previous_response_id=None, prompt=None,
prompt_cache_key=None, reasoning=Reasoning(effort='minimal',
generate_summary=None, summary=None), safety_identifier=None,
service_tier='default', status='completed',
text=ResponseTextConfig(format=ResponseFormatText(type='text'),
verbosity='medium'), top_logprobs=0, truncation='disabled',
usage=ResponseUsage(input_tokens=23,
input_tokens_details=InputTokensDetails(cached_tokens=0), output_tokens=68,
output_tokens_details=OutputTokensDetails(reasoning_tokens=0), total_tokens=91),
user=None, billing={'payer': 'openai'}, store=True)
```

[116]:
```
# Let's only print the textual response part.
print(response1.output[1].content[0].text)
```

```
Antonio Vivaldi. He composed the set of violin concertos known as The Four
Seasons (Le quattro stagioni), published as Op. 8 (RV 269, 315, 293, 297) around
1725.
```

1.2.1 Using the `output_text` convenience property

OpenAI API documentation

[117]:
```
# Another way of printing only the textual part of the response, using the␣
↪`output_text` convenience property.
print(response1.output_text)
```

```
Antonio Vivaldi. He composed the set of violin concertos known as The Four
Seasons (Le quattro stagioni), published as Op. 8 (RV 269, 315, 293, 297) around
1725.
```

1.2.2 Storing the list of messages in a variable

[123]:
```
conversation1 = [
        {
            "role": "developer",
            "content": "You are a helpful music historian."
        },
        {
            "role": "user",
            "content": "Who composed The Four Seasons?"
        }
    ]

response1 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "minimal"},
    max_output_tokens = 300,
    input = conversation
)

output1 = response1.output[1].content[0].text
print(output1)
```

```
The Four Seasons was composed by Antonio Vivaldi. It is a set of four violin
concertos, each representing a season: Spring, Summer, Autumn, and Winter. They
were published in 1725 as part of a larger collection, Op. 8, commonly known as
Op. 8, The Contest Between Harmony and Invention. The concertos are well known
for their vivid programmatic imagery and virtuoso violin writing.
```

[119]:
```
print(response1.output_text)
```

```
Antonio Vivaldi. He wrote The Four Seasons (Le quattro stagioni), a set of four
violin concertos (Spring, Summer, Autumn,
```

1.3 Simple two-turn conversation, version 1

To continue the conversation, the LLM needs to process the not only the new turn from the user, but also the entire previous conversation, including the assistant response.

[124]:
```
conversation2 = conversation1 + [
    {
        "role": "assistant",
        "content": output1
    },
    {
        "role": "user",
        "content": "For whom were most of his compositions written?"
    }
]

conversation2
```

[124]:
```
[{'role': 'developer', 'content': 'You are a helpful music historian.'},
 {'role': 'user', 'content': 'Who composed The Four Seasons?'},
 {'role': 'assistant',
  'content': 'The Four Seasons was composed by Antonio Vivaldi. It is a set of
four violin concertos, each representing a season: Spring, Summer, Autumn, and
Winter. They were published in 1725 as part of a larger collection, Op. 8,
commonly known as Op. 8, The Contest Between Harmony and Invention. The
concertos are well known for their vivid programmatic imagery and virtuoso
violin writing.'},
 {'role': 'user',
  'content': 'For whom were most of his compositions written?'}]
```

[125]:
```
response2 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "minimal"},
    max_output_tokens = 300,
    input = conversation2
)

output2 = response2.output[1].content[0].text
print(output2)
```

```
Most of Antonio Vivaldi's compositions were written for the female performers at
the Ospedale della Pietà, a girls' orphanage in Venice where he worked for many
years. He was employed as a virtuoso violin teacher and composer, and his
works-concertos, sacred vocal music, and operas-were written for the talented
young women there to perform in church and public events. His collaboration with
```

```
the Pietà's orchestra and chorus produced a large portion of his instrumental
concertos and sacred music.
```

1.4 Simple two-turn conversation example, version 2

An easier alternative is to provide the ID of the previous response in the new request. Note the slightly different response, what may be the reason for it?

[128]:
```
user_message2 = [
    {
        "role": "user",
        "content": "For whom were most of his compositions written?"
    }
]
response2 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "minimal"},
    max_output_tokens = 300,
    previous_response_id = response1.id,
    input = user_message2
)
```

[129]:
```
output2 = response2.output_text
print(output2)
```

```
Most of Antonio Vivaldi's compositions were written for and performed at the
public and private music schools and concerts in Venice, especially for the
Ospedale della Pietà, a Venetian orphanage for girls where he worked for many
years. He also wrote extensively for the church and for aristocratic patrons.
So, while he wrote for various patrons, a large portion of his famous
work-including many concertos-originated from his time at the Pietà and from
commissions tied to the Venetian musical scene.
```

Let's extract just the composer's name from the first response. We will supply the JSON schema in the API call, leveraging the Structured Outputs capability.

[130]:
```
user_message3 = [
    {
        "role": "user",
        "content": "Output the composer and the composition."
    }
]
response3 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "low"},
    max_output_tokens = 500,
    previous_response_id = response1.id,
    input = user_message3,
```

```
    text = {
        "format": {
            "type": "json_schema",
            "name": "example_response",
            "schema": {
                "type": "object",
                "properties": {
                    "composer": {"type": "string"},
                    "composition": {"type": "string"}
                },
            "required": ["composer", "composition"],
            "additionalProperties": False
            },
            "strict": True
        }
    }
)
```

[131]:
```
output3 = response3.output_text
print(output3)
```

```
{"composer":"Antonio Vivaldi","composition":"The Four Seasons"}
```

What if we specified Structured Output from the beginning?

[132]:
```
conversation4 = [
        {
            "role": "developer",
            "content": "You are a helpful music historian."
        },
        {
            "role": "user",
            "content": "Who composed The Four Seasons? Give me just the␣
 ↪composer name together with the name of the composition."
        }
    ]

response4 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "low"},
    max_output_tokens = 500,
    input = conversation4,
    text = {
        "format": {
            "type": "json_schema",
            "name": "example_response",
            "schema": {
                "type": "object",
```

```
                "properties": {
                    "composer": {"type": "string"},
                    "composition": {"type": "string"}
                },
            "required": ["composer", "composition"],
            "additionalProperties": False
            },
            "strict": True
        }
    }
)
```

[133]:
```
output4 = response4.output_text
print(output4)
```

```
{"composer":"Antonio Vivaldi","composition":"The Four Seasons"}
```

[ ]:
```

```

1.5 Text style transfer

1.5.1 Changing the narrative perspective

[143]:
```
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."

conversation5 = [
    {"role": "developer", "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}"'}
]

response5 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "low"},
    max_output_tokens = 5000,
```

```
    input = conversation5
)
```

[144]:
```
output5 = response5.output_text
print(output5)
```

Suddenly she could hear Q-Tip, with his human voice, rapping over a human beat. And 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? She 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.

1.5.2 Stream of consciousness

[145]:
```
conversation6 = [
    {"role": "developer", "content": "You are a helpful writer assistant."},
    {"role": "user",
             "content": f'Rewrite this text in a stream of consciousness style:␣
  ↪"{sample_zs}"'}
]

response6 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "low"},
    max_output_tokens = 5000,
    input = conversation6
)
```

[146]:
```
output6 = response6.output_text
print(output6)
```

Suddenly I hear Q-Tip's voice in there, not some movie version but the real, warm human intonation, rapping over a living beat, and there it is-my skull yawning open, top peeled back to admit this human Q-Tip, and across the sea of bodies a rail-thin man with eyes that seem to measure galaxies reaches out, hand snagging mine, keeps asking, again and again, you feeling it? I am feeling it-oh am I feeling it-though the ridiculous heels bite into my feet, a thorny crown of pain, making me fear I might die right here, right now, yet somehow bursting with luck and awe that the song should be playing at this exact moment in the history of the world, as if time itself had paused to tilt its head and listen. I grip the hand offered, and the top of my head bursts free, sky chewing the room as if the crown of me could finally fly.

1.6 Sequence completion

[138]:
```
conversation7 = [
    {"role": "developer", "content": "You are a helpful assistant."},
    {"role": "user",
             "content": 'Hey all mighty GPT, can you tell me what number␣
 ↪follows in this sequence: -2, 4, 3, 7, 8, 10, 13, 13, 18, 16, ...'}]

response7 = client.responses.create(
    model = "gpt-5-nano",
    reasoning = {"effort": "low"},
    max_output_tokens = 500,
    input = conversation7
)
```

[139]:
```
output7 = response7.output_text
print(output7)
```

```
23

Reason: the sequence splits into two interleaved sequences:
- Odd positions: -2, 3, 8, 13, 18, … increase by 5 each time. So the next odd
term (a11) is 18 + 5 = 23.
- Even positions: 4, 7, 10, 13, 16, … increase by 3 each time.

Since the next term after a10 is a11 (an odd position), the next number is 23.
```

1.7 More reasoning examples

1.7.1 Pattern recognition

[140]:
```
examples = "Apple -> Xxxx\n" \
            "Frog -> Xxxx\n" \
            "pop -> xxx\n" \
            "current -> xxx\n" \
            "CNN -> XXX\n" \
            "ABBA -> XXX\n"
test = "Ftp ->"

conversation8 = [
    {"role": "developer", "content": "You are a good at spotting patterns in␣
 ↪text."},
    {"role": "user",
             "content": f'Consider the <input -> output> training exampes below:
 ↪\n{examples}' +
                        f'\nPredict the output for the input given below:
 ↪\n{test}'}]
print(conversation8[1]['content'])
```

```
response8 = client.responses.create(
    model = "gpt-5-mini",
    #reasoning = {"effort": "low"},
    max_output_tokens = 5000,
    tool_choice = 'none',
    input = conversation8
)
```

```
Consider the <input -> output> training exampes below:
Apple -> Xxxx
Frog -> Xxxx
pop -> xxx
current -> xxx
CNN -> XXX
ABBA -> XXX

Predict the output for the input given below:
Ftp ->
```

[141]:
```
output8 = response8.output_text
print(output8)
```

```
Ftp -> Xxx

(Uppercase letters → X, lowercase → x.)
```

[142]:
```
new_test = "Gremlin -> "

response9 = client.responses.create(
    model = "gpt-5-mini",
    #reasoning = {"effort": "low"},
    max_output_tokens = 5000,
    tool_choice = 'none',
    previous_response_id = response8.id,
    input = f'Predict the output for the input given below:\n{new_test}'
)

output9 = response9.output_text
print(output9)
```

```
Gremlin -> Xxxxxxx

(Uppercase letters → X, lowercase → x.)
```

Generalizing from very few examples

[2]:
```
examples = "0 -> 1\n" \
           "1 -> 2\n"
```

```
test = "2 ->"

conversation10 = [
    {"role": "developer", "content": "You are good at learning patterns."},
    {"role": "user",
             "content": f'Consider the <input -> output> training exampes below:
 ↪\n{examples}' +
                         f'\nPredict the output for the input given below:
 ↪\n{test}'}]
print(conversation10[1]['content'])

response10 = client.responses.create(
    model = "gpt-5-nano",
    #reasoning = {"effort": "low"},
    max_output_tokens = 5000,
    tool_choice = 'none',
    input = conversation10
)

output10 = response10.output_text
print(output10)
```

```
Consider the <input -> output> training exampes below:
0 -> 1
1 -> 2

Predict the output for the input given below:
2 ->
3

(The pattern is input + 1: 0→1, 1→2, so 2→3.)
```

1.7.2 Red herring

[3]:
```
examples = "a0 -> 1\n" \
            "f1 -> 2\n" \
            "w0 -> 1\n" \
            "r2 -> 0\n" \
            "h1 -> 2\n"
test = "d2 ->"

conversation11 = [
    {"role": "developer", "content": "You are good at learning patterns."},
    {"role": "user",
             "content": f'Consider the <input -> output> training exampes below:
 ↪\n{examples}' +
```

```
                        f'\nPredict the output for the input given below:
  ↪\n{test}'}]
print(conversation11[1]['content'])

response11 = client.responses.create(
    model = "gpt-5-nano",
    #reasoning = {"effort": "low"},
    max_output_tokens = 5000,
    tool_choice = 'none',
    input = conversation11
)

output11 = response11.output_text
print(output11)
```

```
Consider the <input -> output> training exampes below:
a0 -> 1
f1 -> 2
w0 -> 1
r2 -> 0
h1 -> 2

Predict the output for the input given below:
d2 ->
0

Reason: The given examples suggest the output depends only on the trailing
digit: 0→1, 1→2, 2→0. Since the input ends with 2, the output is 0.
```

1.7.3 Explanations

There are at least 3 ways of obtaining an explanation:

1. Specify that you want an explanation to start with, e.g. “Consider the input …. Predict hte ouput … Provide an explanation”.
2. Use a second turn to ask for an explanation (see below).
3. Use `reasoning = {'summary' : 'detailed', 'effort' : 'medium'}`, then print this reasoning summary.

[4]:
```
response12 = client.responses.create(
    model = "gpt-5-mini",
    #reasoning = {"effort": "low"},
    max_output_tokens = 5000,
    tool_choice = 'none',
    previous_response_id = response11.id,
    input = f'Great! Can you explain the pattern that you learned?'
)
```

```
output12 = response12.output_text
print(output12)
```

```
From the examples the output depends only on the final digit, not the letter.
The mapping is:
- 0 → 1
- 1 → 2
- 2 → 0

This is the same as output = (input digit + 1) mod 3. So for d2 the input digit
is 2, and (2 + 1) mod 3 = 0, hence the output is 0.
```

1.8 Image Understanding

[9]:
```
response13 = client.responses.create(
    model = "gpt-5",
    input = [{
        "role": "user",
        "content": [
            {"type": "input_text", "text": "What's in this image?"},
            {
                "type": "input_image",
                "image_url": "https://webpages.charlotte.edu/rbunescu/courses/
 ↪itcs4101/examples/pump.jpg",
            },
        ],
    }],
)

print(response13.output_text)
```

```
A gas-station pump. The digital screen shows $29.14 and 10.403 gallons. It's
pump number 5 with various stickers and QR codes, including BP/Amoco and a "Save
10¢ a gallon with earnify & Amazon Prime" promo. There are safety notices like
"Do not use phone while refueling," accessibility info, and warning labels. A
person's reflection is visible in the pump display.
```

[7]:
```
response14 = client.responses.create(
    model = "gpt-5-mini",
    tool_choice = "none",
    input = 'What is the price per gallon of fuel that is charged by a pump ' +
            'that has the details shown below?\n' +
            response13.output_text
)

print(response14.output_text)
```

```
Price per gallon = total ÷ gallons = $28.74 ÷ 10.403  $2.763/gal.
So about $2.76 per gallon (rounded).
```

1.8.1 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-..."
```