ExamplesGPT

March 3, 2026

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-5.2-pro`, `gpt-5.2-codex`, `gpt-5`, `gpt-5-mini`, …, or `gpt-5-nano`. For very simple questions, use `gpt-5-nano`, it is cheapest.

[2]:
```
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_0dc5cadda0a8ea6c0069a74286bf988193a1b816903c39bd23',
created_at=1772569222.0, error=None, incomplete_details=None, instructions=None,
metadata={}, model='gpt-5-nano-2025-08-07', object='response', output=[ResponseR
easoningItem(id='rs_0dc5cadda0a8ea6c0069a742871eb481938e7f1e3f7ffb40ef',
summary=[], type='reasoning', content=None, encrypted_content=None,
status=None), ResponseOutputMessage(id='msg_0dc5cadda0a8ea6c0069a742873e34819388
a8fe6bc0172f52', content=[ResponseOutputText(annotations=[], text='There are two
well-known sets commonly referred to as "The Four Seasons":\n\n1) Antonio
Vivaldi’s The Four Seasons (The Four Seasons, Op. 8, a group of four violin
concertos: Spring, Summer, Autumn, Winter), composed in 1720 and published in
1725 as part of the collection Il cimento dell’armonia e dell’inventione.\n\n2)
Sometimes people refer to The Four Seasons by other composers (less common in
classical repertoire), but the iconic and most recognized is Vivaldi’s.\n\nIf
you meant a different work with the same name, tell me and I can clarify.',
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=147,
output_tokens_details=OutputTokensDetails(reasoning_tokens=0),
total_tokens=170), user=None, billing={'payer': 'openai'},
```

completed_at=1772569223, frequency_penalty=0.0, presence_penalty=0.0, prompt_cache_retention=None, store=True)

[3]:
```
# Let's only print the textual response part.
print(response1.output[1].content[0].text)
```

There are two well-known sets commonly referred to as "The Four Seasons":

1) Antonio Vivaldi's The Four Seasons (The Four Seasons, Op. 8, a group of four violin concertos: Spring, Summer, Autumn, Winter), composed in 1720 and published in 1725 as part of the collection Il cimento dell'armonia e dell'inventione.

2) Sometimes people refer to The Four Seasons by other composers (less common in classical repertoire), but the iconic and most recognized is Vivaldi's.

If you meant a different work with the same name, tell me and I can clarify.

1.2.1 Using the output_text convenience property

OpenAI API documentation

[4]:
```
# Another way of printing only the textual part of the response, using the `output_text` convenience property.
print(response1.output_text)
```

There are two well-known sets commonly referred to as "The Four Seasons":

1) Antonio Vivaldi's The Four Seasons (The Four Seasons, Op. 8, a group of four violin concertos: Spring, Summer, Autumn, Winter), composed in 1720 and published in 1725 as part of the collection Il cimento dell'armonia e dell'inventione.

2) Sometimes people refer to The Four Seasons by other composers (less common in classical repertoire), but the iconic and most recognized is Vivaldi's.

If you meant a different work with the same name, tell me and I can clarify.

1.2.2 Storing the list of messages in a variable

[5]:
```
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 = conversation1
)

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 one representing a season: Spring, Summer, Autumn, and Winter.
They were published as Op. 8 in 1725, with the full work often referred to as a
concerto grosso or violin concertos, and they are among the most famous works of
Baroque music. If you'd like, I can list the individual concertos and provide a
bit of historical context or notable recordings.
```

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

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.

1.3.1 Vanilla LLM's have no memory of previous interactions.

In this example, the LLM will knot know who 'his' refers to, hence it return nothing.

[19]:
```
conversationx = [
        {
            "role": "user",
            "content": "For whom were most of his compositions written?"
        }
    ]

responsex = client.responses.create(
    model = "gpt-5-nano",
    max_output_tokens = 300,
    input = conversationx
)

print(responsex.output[0].content)
```

```
None
```

1.3.2 We need to provide the entire conversation history

[8]:
```
conversation2 = conversation1 + [
    {
        "role": "assistant",
        "content": output1
    },
    {
        "role": "user",
        "content": "For whom were most of his compositions written?"
    }
]

conversation2
```

[8]:
```
[{'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 one representing a season: Spring, Summer, Autumn,
and Winter. They were published as Op. 8 in 1725, with the full work often
referred to as a concerto grosso or violin concertos, and they are among the
most famous works of Baroque music. If you'd like, I can list the individual
concertos and provide a bit of historical context or notable recordings.'},
 {'role': 'user',
  'content': 'For whom were most of his compositions written?'}]
```

[9]:
```
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 Vivaldi's compositions were written for the Ospedale della Pietà, an
orphanage in Venice where he worked as a violin teacher and conductor. He also
composed extensively for the girls' choir and string orchestra there, as well as
for other patrons in Venice, including aristocratic and religious institutions.
His role at the Pietà helped him develop a large output of instrumental
concertos, sacred music, and vocal works aimed at the performers and audiences
connected with the hospital. If you'd like, I can summarize his key patrons and
the types of works he wrote for them.
```

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?

[10]:
```
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
)
```

[11]:
```
output2 = response2.output_text
print(output2)
```

```
Most of Antonio Vivaldi's compositions were written for the students and
patients at the Ospedale della Pietà in Venice. This orphanage and its
associated music school housed a large musical establishment where many
performances, training, and premieres took place. Vivaldi was employed there for
much of his career, first as a violin teacher and conductor, later as maestro di
violino and maestro di tutti I concerti (master of all concertos). The
institution provided many violin students, especially girls, who formed the core
performers for his concertos, cantatas, and sacred works.

If you'd like, I can add a brief note on the roles he held at the Pietà and how
that environment influenced his output.
```

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.

[12]:
```
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
        }
    }
)
```

[13]:
```
output3 = response3.output_text
print(output3)
```

```
{"composer":"Antonio Vivaldi","composition":"The Four Seasons (Le quattro
stagioni), Op. 8 - a set of four violin concertos for Spring, Summer, Autumn,
and Winter."}
```

What if we specified Structured Output from the beginning?

[14]:
```
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
        }
    }
)
```

[15]:
```
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

[20]:
```
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
)
```

[21]:
```
output5 = response5.output_text
print(output5)
```

Zadie suddenly 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

[22]:
```
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
)
```

[23]:
```
output6 = response6.output_text
print(output6)
```

Suddenly I hear Q-Tip, yeah, with his human voice sliding into the room, rapping over a human beat, and the top of my skull just cracks open, lets him in, and then this rail-thin man with enormous eyes crawls across a river of bodies to grab my hand, keeps asking me the same question like a rhythm I can't shake: You feeling it? I am, aren't I? My ridiculous heels are stabbing my feet, I'm terrified I might die, and yet there's this wild, dizzy swell of delight that the song should be happening right now, right at this exact moment in the history of the world, as if the universe paused to press play on me. I squeeze his hand, and the top of my head shoots off like a lid, sailing away into whatever sky or memory or dream waits beyond.

1.6 Sequence completion

[24]:
```
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
)
```

[25]:
```
output7 = response7.output_text
print(output7)
```

```
23

Reason:
The sequence splits into two interleaved sequences:
- Odd positions (1,3,5,…): -2, 3, 8, 13, 18,… which increases by 5 each
step. Formula: -2 + 5(n-1).
- Even positions (2,4,6,…): 4, 7, 10, 13, 16,… which increases by 3 each
step. Formula: 4 + 3(n-1).

The next term is the 11th element (odd position), which corresponds to n=6 in
the odd-sequence:
-2 + 5(6-1) = -2 + 25 = 23.
```

1.7 More reasoning examples

1.7.1 Pattern recognition

[26]:
```
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 ->
```

[27]:
```
output8 = response8.output_text
print(output8)
```

```
Xxx

Reason: each run of same case maps to X (uppercase) or x (lowercase) repeated up
to a maximum of 3 characters. F (upper) -> X; "tp" (two lowers) -> xx; combined
-> Xxx.
```

[28]:
```
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)
```

```
Xxxx
```

```
Reason: Uppercase run "G" -> X; lowercase run "remlin" (6 letters) -> xxx
(capped at 3). Combined -> Xxxx.
```

Generalizing from very few examples

[29]:
```
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
```

1.7.2 Red herring

[30]:
```
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 pattern is output = (digit + 1) mod 3, independent of the letter.
For d2, digit = 2 → (2 + 1) mod 3 = 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 the 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.

[31]:
```
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 training pairs you gave, I looked only at the digit on the right and
the corresponding output:

- a0 -> 1  (0 maps to 1)
- f1 -> 2  (1 maps to 2)
- w0 -> 1  (0 maps to 1 again)
- r2 -> 0  (2 maps to 0)
- h1 -> 2  (1 maps to 2 again)

The letter on the left varies with no change to the mapping, so it appears
irrelevant. The digit-to-output mapping is:

- 0 → 1
- 1 → 2
- 2 → 0

That is a cyclic shift: output = (input digit + 1) mod 3. Applying that rule to
d2 gives (2 + 1) mod 3 = 0.
```

1.8 Image Understanding

[32]:
```
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 close-up of a gas pump. The screen shows about $29.14 for 10.403 gallons. It's
pump number 5, with various stickers/QR codes including a "Save 10¢ a gallon
```

```
with earnify & Amazon Prime" ad and a "Do not use phone while refueling"
warning. A faint reflection of the photographer is visible in the display.
```

[33]:
```
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 = $29.14 ÷ 10.403  $2.8011, so about $2.80 per gallon.
```

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-..."
```