

Assistants Code Interpreter Beta

[Copy page](#)

Code Interpreter allows Assistants to write and run Python code in a sandboxed execution environment. This tool can process files with diverse data and formatting, and generate files with data and images of graphs. Code Interpreter allows your Assistant to run code iteratively to solve challenging code and math problems. When your Assistant writes code that fails to run, it can iterate on this code by attempting to run different code until the code execution succeeds.


See a quickstart of how to get started with Code Interpreter [here](#).

How it works

Code Interpreter is charged at \$0.03 per session. If your Assistant calls Code Interpreter simultaneously in two different threads (e.g., one thread per end-user), two Code Interpreter sessions are created. Each session is active by default for one hour, which means that you only pay for one session per if users interact with Code Interpreter in the same thread for up to one hour.

Enabling Code Interpreter


Pass `code_interpreter` in the `tools` parameter of the Assistant object to enable Code Interpreter:

```
python ↕   
1 assistant = client.beta.assistants.create(  
2     instructions="You are a personal math tutor. When asked a math question, write and run code  
3     model="gpt-4o",  
4     tools=[{"type": "code_interpreter"}]  
5 )
```

The model then decides when to invoke Code Interpreter in a Run based on the nature of the user request. This behavior can be promoted by prompting in the Assistant's `instructions` (e.g., “write code to solve this problem”).

Passing files to Code Interpreter

Files that are passed at the Assistant level are accessible by all Runs with this Assistant:

```
python ↕ 
```

```

1 # Upload a file with an "assistants" purpose
2 file = client.files.create(
3     file=open("mydata.csv", "rb"),
4     purpose='assistants'
5 )
6
7 # Create an assistant using the file ID
8 assistant = client.beta.assistants.create(
9     instructions="You are a personal math tutor. When asked a math question, write and run code",
10    model="gpt-4o",
11    tools=[{"type": "code_interpreter"}],
12    tool_resources={
13        "code_interpreter": {
14            "file_ids": [file.id]
15        }
16    }
17 )

```

Files can also be passed at the Thread level. These files are only accessible in the specific Thread. Upload the File using the [File upload](#) endpoint and then pass the File ID as part of the Message creation request:

python ↕ 📄

```

1 thread = client.beta.threads.create(
2     messages=[
3         {
4             "role": "user",
5             "content": "I need to solve the equation `3x + 11 = 14`. Can you help me?",
6             "attachments": [
7                 {
8                     "file_id": file.id,
9                     "tools": [{"type": "code_interpreter"}]
10                }
11            ]
12        }
13    ]
14 )

```

Files have a maximum size of 512 MB. Code Interpreter supports a variety of file formats including `.csv`, `.pdf`, `.json` and many more. More details on the file extensions (and their corresponding MIME-types) supported can be found in the [Supported files](#) section below.

Reading images and files generated by Code Interpreter

Code Interpreter in the API also outputs files, such as generating image diagrams, CSVs, and PDFs. There are two types of files that are generated:

- 1 Images

2 Data files (e.g. a `csv` file with data generated by the Assistant)

When Code Interpreter generates an image, you can look up and download this file in the `file_id` field of the Assistant Message response:

```
1 {
2     "id": "msg_abc123",
3     "object": "thread.message",
4     "created_at": 1698964262,
5     "thread_id": "thread_abc123",
6     "role": "assistant",
7     "content": [
8     {
9         "type": "image_file",
10        "image_file": {
11            "file_id": "file-abc123"
12        }
13    }
14 ]
15 # ...
16 }
```

The file content can then be downloaded by passing the file ID to the Files API:

```
python ↕
1 from openai import OpenAI
2
3 client = OpenAI()
4
5 image_data = client.files.content("file-abc123")
6 image_data_bytes = image_data.read()
7
8 with open("./my-image.png", "wb") as file:
9     file.write(image_data_bytes)
```

When Code Interpreter references a file path (e.g., "Download this csv file"), file paths are listed as annotations. You can convert these annotations into links to download the file:

```
1 {
2     "id": "msg_abc123",
3     "object": "thread.message",
4     "created_at": 1699073585,
5     "thread_id": "thread_abc123",
6     "role": "assistant",
7     "content": [
8     {
9         "type": "text",
10        "text": {
11            "value": "The rows of the CSV file have been shuffled and saved to a new CSV file. Y",
12            "annotations": [
```

```

    {
      "type": "file_path",
      "text": "sandbox:/mnt/data/shuffled_file.csv",
      "start_index": 167,
      "end_index": 202,
      "file_path": {
        "file_id": "file-abc123"
      }
    }
  }
  ...

```

Input and output logs of Code Interpreter

By listing the steps of a Run that called Code Interpreter, you can inspect the code `input` and `outputs` logs of Code Interpreter:

python  

```

1 run_steps = client.beta.threads.runs.steps.list(
2   thread_id=thread.id,
3   run_id=run.id
4 )

```

```

1  {
2    "object": "list",
3    "data": [
4      {
5        "id": "step_abc123",
6        "object": "thread.run.step",
7        "type": "tool_calls",
8        "run_id": "run_abc123",
9        "thread_id": "thread_abc123",
10       "status": "completed",
11       "step_details": {
12         "type": "tool_calls",
13         "tool_calls": [
14           {
15             "type": "code",
16             "code": {
17               "input": "# Calculating 2 + 2\nresult = 2 + 2\nresult",
18               "outputs": [
19                 {
20                   "type": "logs",
21                   "logs": "4"
22                 }
23               ]
24             }
25           }
26         ]
27       }
28     ]
29   }

```

Supported files

FILE FORMAT	MIME TYPE
.c	text/x-c
.cs	text/x-csharp
.cpp	text/x-c++
.csv	text/csv
.doc	application/msword
.docx	application/vnd.openxmlformats-officedocument.wordprocessingml.document
.html	text/html
.java	text/x-java
.json	application/json
.md	text/markdown
.pdf	application/pdf
.php	text/x-php
.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
.py	text/x-python
.py	text/x-script.python
.rb	text/x-ruby
.tex	text/x-tex
.txt	text/plain
.css	text/css
.js	text/javascript
.sh	application/x-sh
.ts	application/typescript
.csv	application/csv
.jpeg	image/jpeg
.jpg	image/jpeg
.gif	image/gif
.pk1	application/octet-stream
.png	image/png

FILE FORMAT	MIME TYPE
.tar	application/x-tar
.xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
.xml	application/xml or "text/xml"
.zip	application/zip