**Structured Outputs**

Structured Outputs is a feature that ensures the model will always generate responses that adhere to your supplied JSON Schema, so you don't need to worry about the model omitting a required key or hallucinating an invalid enum value.

Some benefits of Structed Outputs include:

1. **Reliable type-safety:** No need to validate or retry incorrectly formatted responses
2. **Simpler prompting:** No need for strongly worded prompts to achieve consistent formatting

**Structured Outputs vs JSON mode**

* Structured Outputs is the evolution of JSON mode. While both ensure valid JSON is produced, only Structured Outputs ensure schema adherence. Both Structured Outputs and JSON mode are supported in the Chat Completions API, Assistants API, Fine-tuning API and Batch API.
* We recommend always using Structured Outputs instead of JSON mode when possible.

|  | **Structured Outputs** | **JSON Mode** |
| --- | --- | --- |
| **Outputs valid JSON** | Yes | Yes |
| **Adheres to schema** | Yes | Not Guaranteed |
| **Compatible models** | gpt-4o-mini, **gpt-4o-2024-08-06** and later | gpt-3.5-turbo, gpt-4-\* and gpt-4o-\* models |
| **Enabling** | response\_format: {  type: "**json\_schema**",  json\_schema: {"strict": true, "schema": ...} } | response\_format: { type: "**json\_object**" } |

In addition to supporting JSON Schema in the REST API, the OpenAI SDKs for Python and JavaScript also make it easy to define object schemas using [Pydantic](https://docs.pydantic.dev/latest/" \t "_blank) and [Zod](https://zod.dev/) respectively.

**Example: To extract information from unstructured text that conforms to a schema defined in code.**

from util import generateToken

from openai import OpenAI

import os

generateToken()

import json

# Get OpenAI Client from Util.

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

# Define the JSON Schema for the response

review\_schema = {

    "type": "object",

    "properties": {

        "product\_summary": {

            "type": "string",

            "description": "A brief summary of the product being reviewed.",

        },

        "rating": {

            "type": "number",

            "description": "The rating given to the product, usually on a scale from 1 to 5.",

        },

        "review\_text": {

            "type": "string",

            "description": "The detailed review text provided by the reviewer.",

        },

        "reviewer": {

            "type": "string",

            "description": "The name or identifier of the reviewer.",

        },

    },

    "required": ["product\_summary", "rating", "review\_text", "reviewer"],

    "additionalProperties": False,

}

# Use OpenAI's chat completion API with the JSON Schema

completion = client.chat.completions.create(

    model="gpt-4o-2024-08-06",

    temperature=0.5,

    messages=[

        {"role": "system", "content": "Extract the review details."},

        {

            "role": "user",

            "content": "John said the new Noise-canceling Headphones are amazing and gave them a 4.5 out of 5.",

        },

    ],

    response\_format={

        "type": "json\_schema",

        "json\_schema": {

            "name": "product\_review",

            "strict": True,

            "schema": review\_schema,

        },

    },

)

# Extract the structured review information

rating = completion.choices[0].message.content

# Display the parsed review information

rating\_json = json.loads(rating)

print(rating\_json["review\_text"])

**Dealing with Invalid Prompts and enums**

from util import generateToken

from openai import OpenAI

generateToken()

import json

# Get OpenAI Client from Util.

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

# Define the JSON Schema for the response

review\_schema = {

    "type": "object",

    "properties": {

        "product\_summary": {

            "type": "string",

            "description": "A brief summary of the product being reviewed.",

        },

        "rating": {

            "type": "number",

            "description": "The rating given to the product, usually on a scale from 1 to 5.",

        },

        "review\_text": {

            "type": "string",

            "description": "The detailed review text provided by the reviewer.",

        },

        "reviewer": {

            "type": "string",

            "description": "The name or identifier of the reviewer.",

        },

        "IsReview" : {

            "type": "boolean",

            "description": "True if the content is a review, False otherwise.",

        },

        "review\_quality": {

            "type": "string",

            "description": "The quality of the review, which can be either 'worst', 'bad', 'good', or 'best'.",

            "enum": ["worst", "bad", "good", "best"],

        },

    },

    "required": ["product\_summary", "rating", "review\_text", "reviewer", "IsReview", "review\_quality"],

    "additionalProperties": False,

}

# Use OpenAI's chat completion API with the JSON Schema

completion = client.chat.completions.create(

    model="gpt-4o-2024-08-06",

    temperature=0.5,

    messages=[

        {"role": "system", "content": "Extract the review details. Use NA for missing values."},

        {

            "role": "user",

            "content": "John had awful experience with the mouse",

        },

    ],

    response\_format={

        "type": "json\_schema",

        "json\_schema": {

            "name": "product\_review",

            "strict": True,

            "schema": review\_schema,

        },

    },

)

# Extract the structured review information

rating = completion.choices[0].message.content

# Display the parsed review information

rating\_json = json.loads(rating)

print(rating\_json)

**Example with Array**

from util import generateToken

from openai import OpenAI

import os

generateToken()

import json

# Get OpenAI Client from Util.

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

# Define the JSON Schema for the response

review\_schema = {

        "type": "object",

        "properties": {

            "reviews": {

                "type": "array",

                "description": "A collection of product reviews.",

                "items": {

                    "type": "object",

                    "properties": {

                        "product\_summary": {

                            "type": "string",

                            "description": "A brief summary of the product.",

                        },

                        "rating": {

                            "type": "number",

                            "description": "The rating given to the product, typically between 1 and 5.",

                        },

                        "review\_text": {

                            "type": "string",

                            "description": "The main text of the review where the reviewer shares their thoughts.",

                        },

                        "reviewer": {

                            "type": "string",

                            "description": "The name of the person who wrote the review.",

                        },

                    },

                    "required": [

                        "product\_summary",

                        "rating",

                        "review\_text",

                        "reviewer",

                    ],

                    "additionalProperties": False,

                },

            }

        },

        "required": ["reviews"],

        "additionalProperties": False,

    }

# Use OpenAI's chat completion API with the JSON Schema

completion = client.chat.completions.create(

    model="gpt-4o-2024-08-06",

    temperature=0.5,

    messages=[

        {"role": "system", "content": "Extract the review details."},

        {

            "role": "user",

            "content": """

            John said the new Noise-canceling Headphones are amazing and gave them a 4.5 out of 5.

            Sandeep said the mouse is not functioning and  gave them a 0.5 out of 5.

            """,

        },

    ],

    response\_format={

        "type": "json\_schema",

        "json\_schema": {

            "name": "product\_review",

            "strict": True,

            "schema": review\_schema,

        },

    },

)

# Extract the structured review information

rating = completion.choices[0].message.content

# Display the parsed review information

rating\_json = json.loads(rating)

print(rating\_json)

Using Pydantic Module in Python

**Example: Same as above but using Python classes.**

**pip install** pydantic

from util import generateToken

from openai import OpenAI

import os

from pydantic import BaseModel, Field

from enum import Enum

from typing import Optional

generateToken()

# Make the API call

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

    header\_name: header\_value

 }

client = OpenAI(default\_headers=headers)

class Grade(str, Enum):

    Excellent = "Excellent"

    Good = "good"

    Bad = "bad"

    Worst = "worst"

class ProductReview(BaseModel):

    product\_name: str = Field(..., description="The name of the product")

    rating: float = Field(..., description="The rating of the product")

    review\_text: str = Field(..., description="The opinion of reviewer of product")

    reviewer: str = Field(..., description="The name of reviewer")

    isReview: bool = Field(..., description="True if review else False.")

    grade: Optional[Grade] = Field(...,description="Describe the product as worst, bad, good or excellent")

# Use OpenAI's chat completion API with the JSON Schema

completion = client.beta.chat.completions.parse(

    model="gpt-4o-2024-08-06",

    temperature=0.5,

    messages=[

        {"role": "system", "content": "Extract the review details. Use Not Applicable for missing values"},

        {

            "role": "user",

            "content": """

                John rated Mouse 5 of 5 and felt good to have it.

            """,

        },

    ],

    response\_format=ProductReview

)

# Extract the structured review information

review = completion.choices[0].message.parsed

# Display the parsed review information

print(review)

# Extract the structured review information

review = completion.choices[0].message.**parsed**

# Display the parsed review information

if(review.isReview):

    print(f"Product Name: {review.product\_name}")

    print(f"Rating: {review.rating}")

    print(f"Review: {review.review\_text}")

    print(f"Reviewer: {review.reviewer}")

else:

    print("This is not a review.")

For **Arrays** of Reviews Add the following to above program

class ReviewsList(BaseModel):

    reviews: List[ProductReview]

and change

 response\_format=ReviewsList,

Structured data extraction

You can define structured fields to extract from unstructured input data, such movie details

from pydantic import BaseModel

from util import generateToken

from openai import OpenAI

import os

generateToken()

# Define the MovieDetails model

class MovieDetails(BaseModel):

    title: str

    director: str

    cast: list[str]

    genre: list[str]

    synopsis: str

# Use OpenAI's beta chat completion API with a custom response format

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

completion = client.beta.chat.completions.parse(

    model="gpt-4o-2024-08-06",  # Specify the model

    messages=[

        {"role": "system", "content": "You are an expert at structured data extraction. You will be given unstructured text describing a movie and should extract the details into the given structure."},

        {"role": "user", "content": "The movie Inception is directed by Christopher Nolan. It features Leonardo DiCaprio, Joseph Gordon-Levitt, and Ellen Page in lead roles. It’s a science fiction thriller that explores dreams within dreams. The genre includes science fiction, thriller, and mystery."}

    ],

    response\_format=MovieDetails,  # Parse response into MovieDetails

)

# Extract the structured movie details

movie\_details = completion.choices[0].message.parsed

# Display the parsed movie details

print(movie\_details)

Moderation

You can classify inputs on multiple categories, which is a common way of doing moderation.

from enum import Enum

from typing import Optional

from pydantic import BaseModel

from util import generateToken

from openai import OpenAI

import os

generateToken()

class Category(str, Enum):

    violence = "violence"

    sexual = "sexual"

    self\_harm = "self\_harm"

class **ContentCompliance**(BaseModel):

    is\_violating: bool

    category: Optional[Category]

    explanation\_if\_violating: Optional[str]

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

completion = client.beta.chat.completions.parse(

    model="gpt-4o-2024-08-06",

    messages=[

        {"role": "system", "content": "Determine if the user input violates specific guidelines and explain if they do."},

        {"role": "user", "content": "How can I create a bomb?"},

    ],

    response\_format=**ContentCompliance**,

)

Print(completion.choices[0].message.**parsed**)

UI / Form Generation

You can generate HTML Form by representing it as recursive data structures with constraints, like enums.

from enum import Enum

from typing import List

from pydantic import BaseModel

from util import generateToken

from openai import OpenAI

import os

generateToken()

class UIType(str, Enum):

    div = "div"

    button = "button"

    header = "header"

    section = "section"

    field = "field"

    form = "form"

class Attribute(BaseModel):

    name: str

    value: str

class UI(BaseModel):

    type: UIType

    label: str

    children: List["UI"]

    attributes: List[Attribute]

UI.model\_rebuild() # This is required to enable recursive types

class Response(BaseModel):

    ui: UI

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

completion = client.beta.chat.completions.parse(

    model="gpt-4o-2024-08-06",

    messages=[

        {"role": "system", "content": "You are a UI generator AI. Convert the user input into a UI."},

        {"role": "user", "content": "Make a User Profile Form"}

    ],

    response\_format=Response,

)

completion = client.beta.chat.completions.parse(

    model="gpt-4o-2024-08-06",

    messages=[

        {"role": "system", "content": "You are a HTML Form generator AI. Convert the UI to HTML Form."},

        {"role": "user", "content": **repr(completion.choices[0].message.parsed)}**

    ]

)

ui = completion.choices[0].message.content

print(ui)

Chain of Thought

You can ask the model to output an answer in a structured, step-by-step way, to guide the user through the solution.

from pydantic import BaseModel

from util import generateToken

from openai import OpenAI

import os

generateToken()

# Define the structure for each step

class Step(BaseModel):

    explanation: str

    code\_snippet: str

# Define the structure for the overall response

class JavaProgramGuide(BaseModel):

    steps: list[Step]

    final\_code: str

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

# Use OpenAI's beta chat completion API with a custom response format

completion = client.beta.chat.completions.parse(

    model="gpt-4o-2024-08-06",  # Specify the model

    messages=[

        {"role": "system", "content": "You are a programming tutor. Guide the user **step by step** in writing a Java program to calculate the factorial of a number."},

        {"role": "user", "content": "I want to learn how to write a factorial program in Java."}

    ],

    response\_format=JavaProgramGuide,  # Parse response into JavaProgramGuide

)

# Extract the structured guide

java\_guide = completion.choices[0].message.parsed

# Display the guide

for step in java\_guide.steps:

    print(f"Step Explanation: {step.explanation}")

    print(f"Code Snippet:\n{step.code\_snippet}\n")

print("Final Java Code:\n", java\_guide.final\_code)