# Module 9.2 APIs

SPRING 2025 CSD325 ADVANCED PYTHON

**Author**: Brittaney Perry-Morgan
**Date**: July 6th, 2025

## Module 9.2 APIs

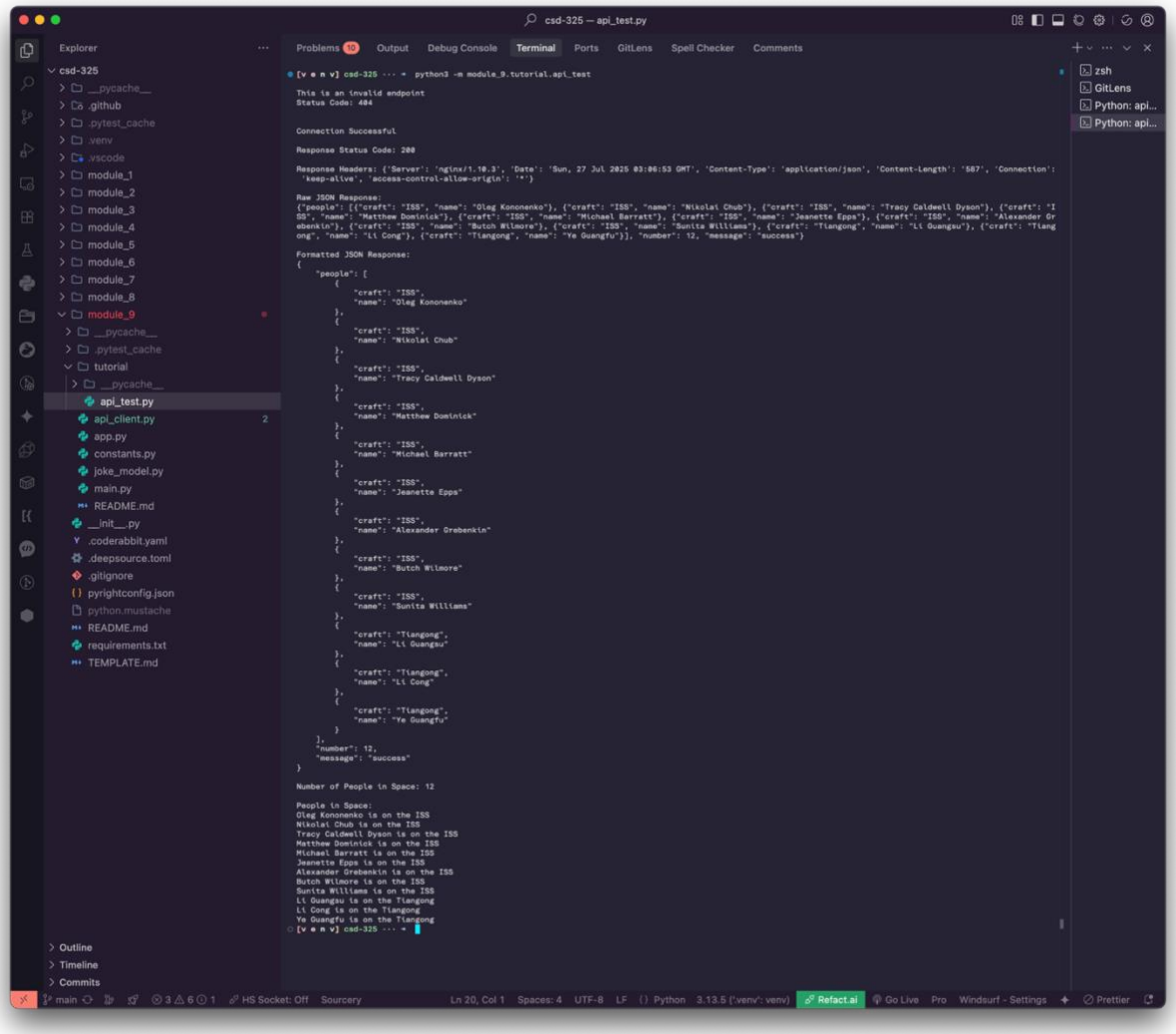
**tutorial/api_test.py**

```python
"""
Name: Brittaney Perry-Morgan
Date: Sunday, July 6th, 2025
Assignment: Module 9.2 API

Purpose: Complete the API tutorial (https://www.dataquest.io/blog/api-in-python/)

Imports:
    - json: Used to format JSON responses.
    - List: Used to define a list of astronauts.
    - TypedDict: Used to define a dictionary of astronauts.
    - requests: Used to make HTTP requests.
"""

import json
from typing import List, TypedDict

import requests


class AstronautInfo(TypedDict):
    """
    Defines the structure for the 'people' part of the API response.

    Fields:
        - name: The name of the astronaut.
        :type name: str

        - craft: The name of the spacecraft the astronaut is in.
        :type craft: str
    """

    name: str
    craft: str


class AstronautsData(TypedDict):
    """
    Defines the overall structure of the astronauts API response.

    Fields:
```

```
        - number: The number of astronauts in space.
        :type number: int

        - people: A list of astronauts in space.
        :type people: List[AstronautInfo]
    """

    number: int
    people: List[AstronautInfo]


def test_open_notify_astronauts_invalid() -> int:
    """
    Test the OpenNotify astronauts API with an invalid endpoint.

    Returns:
        - HTTP status code from the invalid endpoint request.
        :rtype: int

    Raises:
        - requests.exceptions.RequestException: If the request fails completely.
    """
    url = "http://api.open-notify.org/this-api-doesnt-exist"
    try:
        response = requests.get(url, timeout=30)
        return response.status_code
    except requests.exceptions.RequestException as error:
        print(f"Request failed: {error}")
        raise


def test_open_notify_astronauts() -> None:
    """
    Test the OpenNotify astronauts API endpoint and display astronaut information.

    Makes a GET request to the astronauts API, parses the JSON response,
    and prints detailed information about people currently in space.

    Raises:
        - requests.exceptions.RequestException: If the API request fails.
    """
    url = "http://api.open-notify.org/astros"

    try:
        response = requests.get(url, timeout=30)
        response.raise_for_status()  # Raises HTTPError if the request returns unsuccessfully.
        # Cast the JSON response to our defined TypedDict for type safety
        json_data: AstronautsData = response.json()  # Parse JSON once and reuse

        print("\nConnection Successful")
```

```
        print(f"\nResponse Status Code: {response.status_code}")
        print(f"\nResponse Headers: {response.headers}")
        print("\nRaw JSON Response:")
        print(f"{response.text}")
        print("\nFormatted JSON Response:")
        print(f"{format_json_response(json_data)}")
        print(f"\nNumber of People in Space: {json_data['number']}")
        print("\nPeople in Space:")
        for person in json_data["people"]:
            print(f"{person['name']} is on the {person['craft']}")

    except requests.exceptions.RequestException as error:
        print(f"\nError Connecting:\n{error}")


def format_json_response(json_obj: AstronautsData) -> str:
    """
    Format a JSON object (aka the API response).

    Parameters:
        - json_obj: The API response in JSON format.
        :type json_obj: AstronautsData

    Returns:
        - A string representation of the JSON response.
        :rtype: str
    """
    return json.dumps(json_obj, indent=4)


if __name__ == "__main__":
    print(
        f"\nThis is an invalid endpoint\nStatus Code: {test_open_notify_astronauts_invalid()}\n"
    )
    test_open_notify_astronauts()
```

**tutorial/api_test.py (deliverables)**




**constants.py**

```
"""Constant values used throughout the application."""

BASE_URL = "https://v2.jokeapi.dev/joke/"
CATEGORIES = ["Programming", "Misc", "Dark", "Pun", "Spooky", "Christmas"]
JOKE_TYPES = ["single", "twopart"]
MAX_JOKES = 10
```

**joke_model.py**

```
"""
Representation of a joke.

This module holds the Joke dataclass that's used to represent a joke from the API. The dataclass
defines the data structure for a joke.

Imports:
    - json: Used to parse the JSON.
```

```
    - dataclasses: Used to create dataclasses.
    - asdict: Used to convert the dataclass to a dictionary.
    - Optional: Used to type hint optional parameters.
"""

import json
from dataclasses import dataclass, asdict
from typing import Optional


@dataclass
class Joke:
    """
    Representation of a single joke.

    Attributes:
        category: The category of the joke
        joke_type: The type of joke (single or twopart)
        joke_content: The complete joke text for single-type jokes
        setup: The setup line for two-part jokes
        delivery: The punchline for two-part jokes
        raw_json: The original JSON response from the API
    """

    category: str
    joke_type: str
    joke_content: Optional[str] = None
    setup: Optional[str] = None
    delivery: Optional[str] = None
    raw_json: Optional[str] = None

    def __str__(self) -> str:
        """
        Return a string representation of the joke.

        Returns:
            - A string representation of the joke.
            :rtype: str
        """
        return f"{self.category}: {self.joke_content}"

    def print_no_formatting(self) -> None:
        """
        Print the joke in its raw, unformatted form.
        """
        if self.joke_type == "single":
            print(self.joke_content or "No joke content available")
        else:
            print(self.setup or "No setup available")
            print(self.delivery or "No delivery available")
```

```
    def print_formatted_joke(self) -> None:
        """
        Print the joke in a formatted JSON style.

        Returns:
            - None
        """
        # Create a dictionary from the dataclass, excluding the raw_json field
        joke_dict = asdict(self)
        joke_dict.pop("raw_json", None)
        print(json.dumps(joke_dict, indent=4, sort_keys=True))

    def print_raw_json(self) -> None:
        """
        Print the raw JSON response for the joke.

        Returns:
            - None
        """
        print(self.raw_json)
```

**api_client.py**

```
"""
The data access layer of the application.

This module is responsible for the communication with the JokeAPI.

Imports:
    - dataclass: Used to create dataclasses.
    - Optional: Used to type hint optional parameters.
    - TypedDict: Used to define a dictionary of jokes.
    - cast: Used to cast the response to a dictionary.
    - requests: Used to make HTTP requests.
    - BASE_URL: The base URL of the JokeAPI.
    - MAX_JOKES: The maximum number of jokes to retrieve.
"""

from dataclasses import dataclass
from typing import Optional, TypedDict, cast
import requests
from module_9.constants import BASE_URL, MAX_JOKES


class JokeDict(TypedDict):
    """A dictionary representing a single joke from the API."""

    error: bool
    category: str
```

```
    type: str
    joke: Optional[str]
    setup: Optional[str]
    delivery: Optional[str]
    flags: dict[str, bool]
    id: int
    safe: bool
    lang: str


class JokeListResponse(TypedDict):
    """A dictionary representing the response for multiple jokes."""

    error: bool
    amount: int
    jokes: list[JokeDict]


class JokeAPI:
    """
    The client for interacting with the JokeAPI.

    Fields:
        - session: The session used to make HTTP requests.
        :type session: requests.Session
    """

    session: requests.Session

    def __init__(self) -> None:
        """Initialize the client."""
        self.session = requests.Session()
        self.session.headers.update({"Accept": "application/json"})

    def close(self) -> None:
        """Close the HTTP session."""
        self.session.close()

    def __enter__(self):
        """Context manager entry."""
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        """Context manager exit."""
        self.close()

    def get_jokes(
        self, category: str, joke_type: str, amount: int = MAX_JOKES
    ) -> Optional[list[JokeDict]]:
        """
```

```
    Get jokes from the JokeAPI.

    Parameters:
        - category: The category of the joke.
        :type category: str

        - joke_type: The type of joke.
        :type joke_type: str

        - amount: The number of jokes to retrieve.
        :type amount: int

    Returns:
        - A list of jokes if successful, None otherwise.
        :rtype: Optional[list[JokeDict]]
    """
    params = {"type": joke_type, "amount": amount}
    url = f"{BASE_URL}{category}"

    try:
        return self._extracted_from_get_jokes_25(url, params, amount)
    except requests.RequestException as error:
        print(f"Network Error: Could Not Fetch Joke Data:\n{error}")
        return None

def _extracted_from_get_jokes_25(
    self, url: str, params: dict[str, str], amount: int
) -> Optional[list[JokeDict]]:
    """
    Make a GET request to the JokeAPI and return the response.

    Parameters:
        - url: The URL to make the request to.
        :type url: str

        - params: The parameters to pass to the request.
        :type params: dict[str, str]

        - amount: The number of jokes to retrieve.
        :type amount: int

    Returns:
        - A list of jokes if successful, None otherwise.
        :rtype: Optional[list[JokeDict]]
    """
    response = self.session.get(url, params=params, timeout=15)
    response.raise_for_status()
    if amount > 1:
        data = cast(JokeListResponse, response.json())
        if data["error"]:
```

```
            print("API Error: Could not fetch jokes.")
            return None
        return data["jokes"]

    data = cast(JokeDict, response.json())
    if data["error"]:
        print("API Error: Could not fetch joke.")
        return None
    return [data]
```

**app.py**

```
"""
Manage the user's interface and application logic.

Imports:
    - Joke: The Joke dataclass.
    - JokeAPI: The JokeAPI client.
    - CATEGORIES: The available categories for jokes.
    - JOKE_TYPES: The available joke types.
    - MAX_JOKES: The maximum number of jokes to return from the API call.
"""

from module_9.joke_model import Joke
import json
from module_9.api_client import JokeAPI, JokeDict
from module_9.constants import CATEGORIES, JOKE_TYPES


class JokeApp:
    """
    The main application class.

    Fields:
        - api_client: The JokeAPI client.
        :type api_client: JokeAPI
    """

    def __init__(self, api_client: JokeAPI) -> None:
        """
        Initialize the JokeApp.

        Parameters:
            - api_client: The JokeAPI client.
            :type api_client: JokeAPI
        """
        self.api_client = api_client

    def _get_validated_input(
        self, prompt: str, choices: list, default: str | None
```

```
) -> str:
    """
    A private, reusable utility to get a validated choice from the user.

    Parameters:
        prompt: The prompt to display to the user.
        choices: Available choices for the user to choose from.
        default: The default choice if user enters nothing.
    """
    print(f"\n{prompt} (Choices: {', '.join(choices)})")
    while True:
        user_input = input(f"Enter Choice: (Default: {default}): ").strip().lower()
        if not user_input:
            if default:
                return default.lower()
            print("No default available. Please enter a choice.")
            continue
        if user_input in [choice.lower() for choice in choices]:
            return user_input
        print("Invalid Choice. Please Try Again.")

def _map_to_jokes(self, jokes_data: list[JokeDict]) -> list[Joke]:
    """
    Map the raw API data to a list of Joke objects.

    Parameters:
        - jokes_data: The raw API data.
        :type jokes_data: list[JokeDict]

    Returns:
        - A list of Joke objects.
        :rtype: list[Joke]
    """
    return [
        Joke(
            category=joke["category"],
            joke_type=joke["type"],
            joke_content=joke.get("joke"),
            setup=joke.get("setup"),
            delivery=joke.get("delivery"),
            raw_json=json.dumps(joke),
        )
        for joke in jokes_data
    ]

def _display_jokes(self, jokes: list[Joke]) -> None:
    """
    Display the jokes to the user.

    Parameters:
```

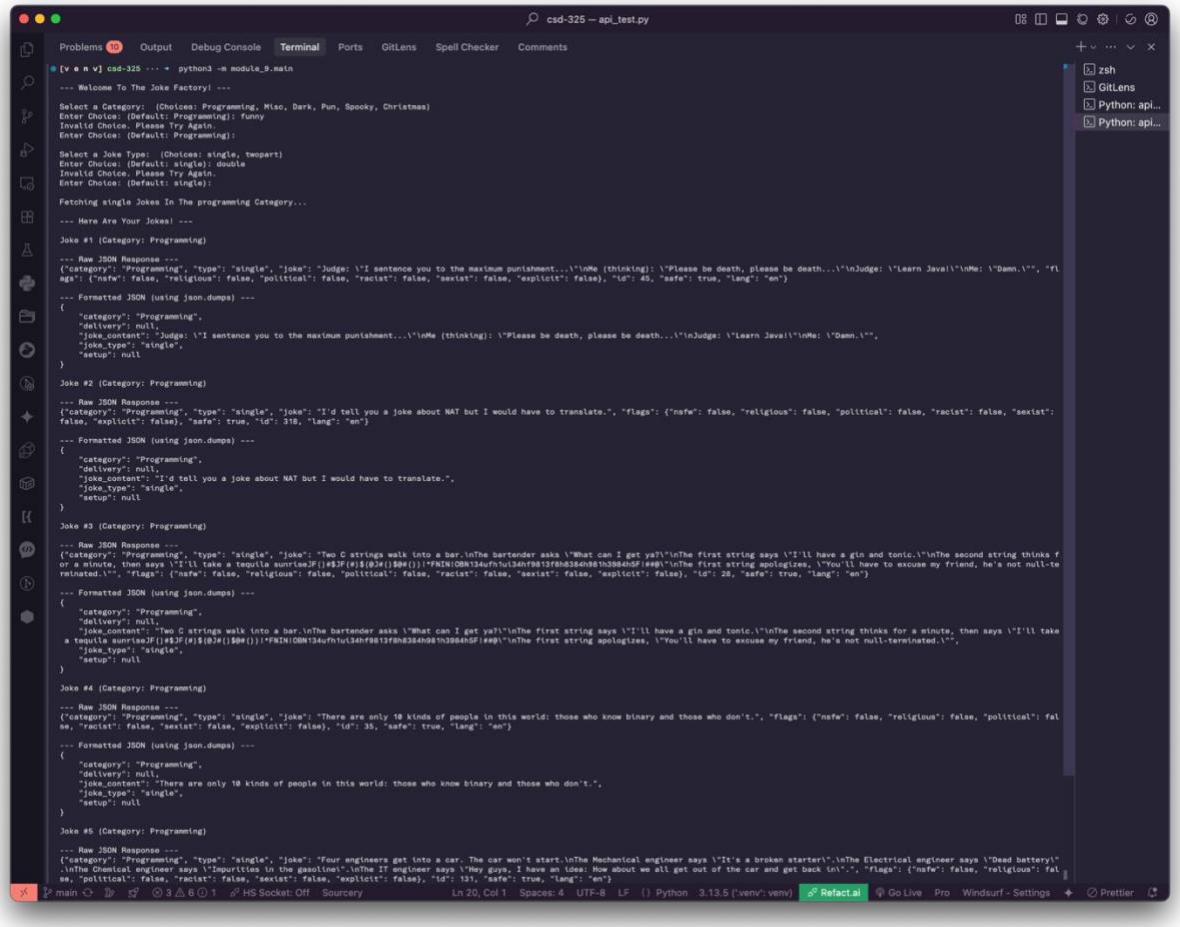
```
            - jokes: The jokes to display.
            :type jokes: list[Joke]
        """
        if not jokes:
            print("\nNo Jokes Found For The Selected Criteria.")
            return

        print("\n--- Here Are Your Jokes! ---")
        for index, joke in enumerate(jokes, 1):
            print(f"\nJoke #{index} (Category: {joke.category})")

            print("\n--- Raw JSON Response ---")
            joke.print_raw_json()

            print("\n--- Formatted JSON (using json.dumps) ---")
            joke.print_formatted_joke()
        print("\n---------------------------")

    def run(self) -> None:
        """
        Run the main application loop.

        Returns:
            - None
        """
        print("\n--- Welcome To The Joke Factory! ---")

        category = self._get_validated_input(
            prompt="Select a Category: ",
            choices=CATEGORIES,
            default="Programming",
        )
        joke_type = self._get_validated_input(
            prompt="Select a Joke Type: ", choices=JOKE_TYPES, default="single"
        )

        print(f"\nFetching {joke_type} Jokes In The {category} Category...")
        if jokes_data := self.api_client.get_jokes(category, joke_type, amount=5):
            jokes = self._map_to_jokes(jokes_data)
            self._display_jokes(jokes)
```

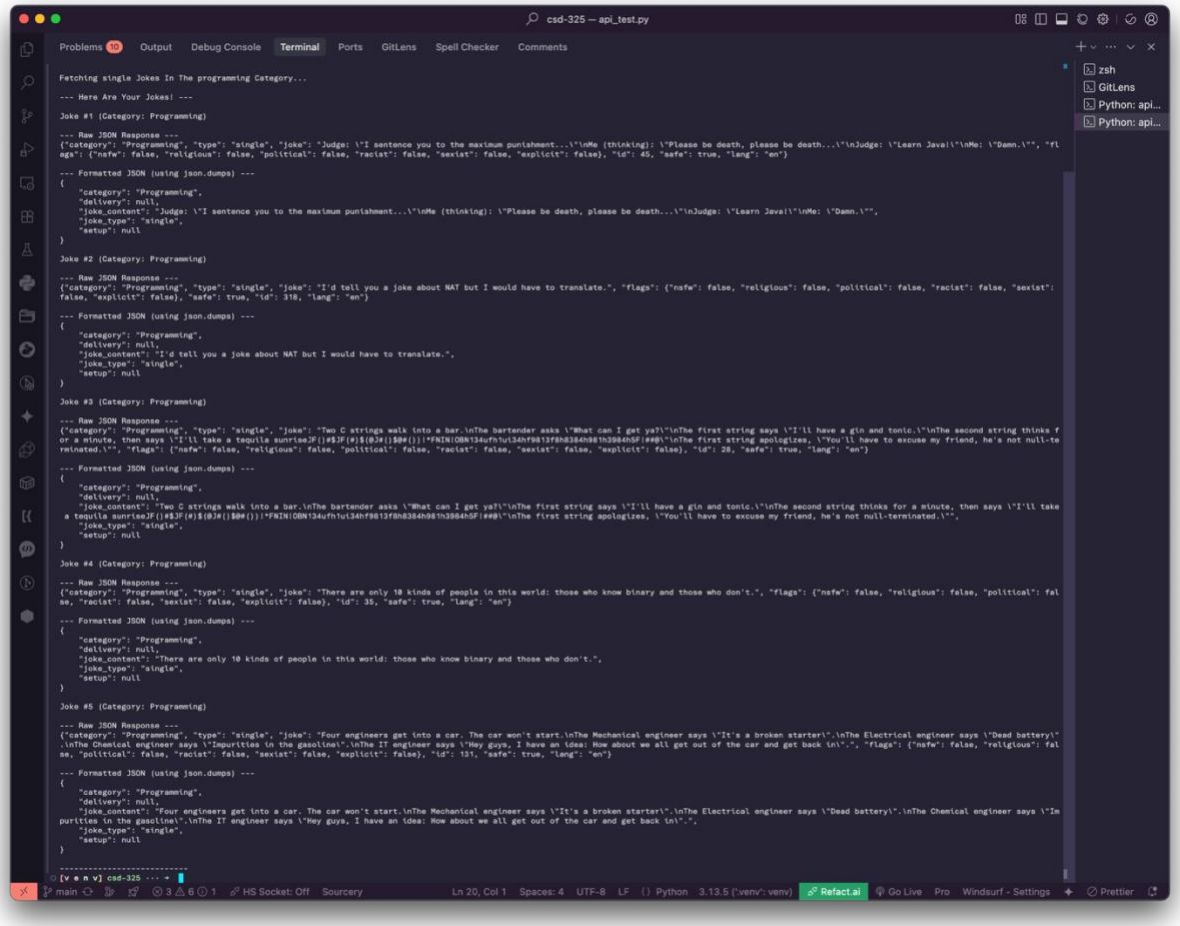
**main.py**

```
"""
Name: Brittaney Perry-Morgan
Date: Sunday, July 6th, 2025
Assignment: Module 9.2 API

Purpose: Demonstrate the ability to interact with a public API (https://sv443.net/jokeapi/v2/).
```

```python
    This is the main entry point for the application.
    """

from module_9.app import JokeApp
from module_9.api_client import JokeAPI


def main():
    """The main function of the application."""
    api_client = JokeAPI()
    app = JokeApp(api_client=api_client)
    app.run()


if __name__ == "__main__":
    main()
```

**deliverables**




csd-325 — api_test.py
Problems 10
Output
Debug Console
Terminal
Ports
GitLens
Spell Checker
Comments
zsh
GitLens
Python: api...
Python: api...
Fetching single Jokes In The programming Category...
--- Here Are Your Jokes! ---
Joke #1 (Category: Programming)
--- Raw JSON Response ---
{"category": "Programming", "type": "single", "joke": "Judge: \"I sentence you to the maximum punishment...\"\nMe (thinking): \"Please be death, please be death...\"\nJudge: \"Learn Java!\"\nMe: \"Damn.\"", "flags": {"nsfw": false, "religious": false, "political": false, "racist": false, "sexist": false, "explicit": false}, "id": 45, "safe": true, "lang": "en"}
--- Formatted JSON (using json.dumps) ---
{
"category": "Programming",
"delivery": null,
"joke_content": "Judge: \"I sentence you to the maximum punishment...\"\nMe (thinking): \"Please be death, please be death...\"\nJudge: \"Learn Java!\"\nMe: \"Damn.\"",
"joke_type": "single",
"setup": null
}
Joke #2 (Category: Programming)
--- Raw JSON Response ---
{"category": "Programming", "type": "single", "joke": "I'd tell you a joke about NAT but I would have to translate.", "flags": {"nsfw": false, "religious": false, "political": false, "racist": false, "sexist": false, "explicit": false}, "safe": true, "id": 318, "lang": "en"}
--- Formatted JSON (using json.dumps) ---
{
"category": "Programming",
"delivery": null,
"joke_content": "I'd tell you a joke about NAT but I would have to translate.",
"joke_type": "single",
"setup": null
}
Joke #3 (Category: Programming)
--- Raw JSON Response ---
{"category": "Programming", "type": "single", "joke": "Two C strings walk into a bar.\nThe bartender asks \"What can I get ya?\"\nThe first string says \"I'll have a gin and tonic.\"\nThe second string thinks for a minute, then says \"I'll take a tequila sunrise\nJF()#$JF(#)$(@J#()$@#())!*FNIN!OBN134ufh1ui34hf9813f8h8384h981h3984hSF|##@\"\nThe first string apologizes, \"You'll have to excuse my friend, he's not null-terminated.\"", "flags": {"nsfw": false, "religious": false, "political": false, "racist": false, "sexist": false, "explicit": false}, "id": 28, "safe": true, "lang": "en"}
--- Formatted JSON (using json.dumps) ---
{
"category": "Programming",
"delivery": null,
"joke_content": "Two C strings walk into a bar.\nThe bartender asks \"What can I get ya?\"\nThe first string says \"I'll have a gin and tonic.\"\nThe second string thinks for a minute, then says \"I'll take a tequila sunrise\nJF()#$JF(#)$(@J#()$@#())!*FNIN!OBN134ufh1ui34hf9813f8h8384h981h3984hSF|##@\"\nThe first string apologizes, \"You'll have to excuse my friend, he's not null-terminated.\"",
"joke_type": "single",
"setup": null
}
Joke #4 (Category: Programming)
--- Raw JSON Response ---
{"category": "Programming", "type": "single", "joke": "There are only 10 kinds of people in this world: those who know binary and those who don't.", "flags": {"nsfw": false, "religious": false, "political": false, "racist": false, "sexist": false, "explicit": false}, "id": 35, "safe": true, "lang": "en"}
--- Formatted JSON (using json.dumps) ---
{
"category": "Programming",
"delivery": null,
"joke_content": "There are only 10 kinds of people in this world: those who know binary and those who don't.",
"joke_type": "single",
"setup": null
}
Joke #5 (Category: Programming)
--- Raw JSON Response ---
{"category": "Programming", "type": "single", "joke": "Four engineers get into a car. The car won't start.\nThe Mechanical engineer says \"It's a broken starter\".\nThe Electrical engineer says \"Dead battery\".\nThe Chemical engineer says \"Impurities in the gasoline\".\nThe IT engineer says \"Hey guys, I have an idea: How about we all get out of the car and get back in\".", "flags": {"nsfw": false, "religious": false, "political": false, "racist": false, "sexist": false, "explicit": false}, "id": 131, "safe": true, "lang": "en"}
--- Formatted JSON (using json.dumps) ---
{
"category": "Programming",
"delivery": null,
"joke_content": "Four engineers get into a car. The car won't start.\nThe Mechanical engineer says \"It's a broken starter\".\nThe Electrical engineer says \"Dead battery\".\nThe Chemical engineer says \"Impurities in the gasoline\".\nThe IT engineer says \"Hey guys, I have an idea: How about we all get out of the car and get back in\".",
"joke_type": "single",
"setup": null
}
---
(venv) csd-325
main
HS Socket: Off
Sourcery
Ln 20, Col 1
Spaces: 4
UTF-8
LF
Python
3.13.5 ('venv': venv)
Refact.ai
Go Live
Pro
Windsurf - Settings
Prettier