HTTP-Python Response_object_ details

Introduction

When using the requests library in Python to send a GET request, you receive a Response object, which contains various attributes and methods providing comprehensive information about the response. Here's an explanation of each attribute and method available in the Response object:

1. response.status_code

Description: Provides the HTTP Status Code indicating the request's status. For example:

- 200: Success
- 404: Resource not found
- 500: Server error

Example:

```
print(response.status_code)
```

2. response.headers

Description: Displays the headers sent with the response, which contain additional information from the server.

Example:

```
print(response.headers)
```

Example Output:

```
{'Content-Type': 'application/json', 'Content-Length': '123', ...}
```

3. response.content

Description: Provides the entire response content in binary format, typically used for non-text data like images or files.

Example:

```
print(response.content)
```

4. response.text

Description: Returns the response content as text, automatically decoding it using the appropriate encoding.

Example:

```
print(response.text)
```

5. response.json()

- **Description**: Converts the response content to a JSON object if the data is in JSON format.

Example:

```
data = response.json()
print(data)
```

Note: If the data is not in JSON format, this will raise an exception.

response.url

Description: Displays the final URL of the request, especially useful if there was a redirection.

Example:

```
print(response.url)
```

7. response.encoding

Description: Displays or sets the encoding used to convert response.content to response.text. Encoding is detected automatically but can be set manually.

Example:

```
print(response.encoding)
```

```
response.encoding = 'utf-8'
```

10. response.raise_for_status()

Description: Raises an exception if the response contains an error status code (like 4xx or 5xx).

Example:

```
try:
    response.raise_for_status()
except requests.exceptions.HTTPError as e:
   print(f"HTTP error occurred: {e}")
```

11. response.elapsed

Description: Shows the time taken to complete the request as a timedelta object.

Example:

```
print(response.elapsed)
```

12. response.ok

Description: Returns True if the status code indicates success (from 200 to 299).

Example:

```
print(response.ok)
```

13. response.apparent_encoding

Description: Shows the apparent encoding based on the response content, useful if encoding is not specified.

Example:

```
print(response.apparent_encoding)
```

14. response.links

Description: Displays any additional links provided in the response (such as pagination links in paginated APIs).

Example:

```
print(response.links)
```

15. response.close()

Description: Closes the response connection. Responses are closed automatically, but you can use close() for manual connection management.

Example:

```
response.close()
```

16. response.iter_content(chunk_size=1024)

Description: Used to read the response content in chunks of a specified byte size. Useful for handling large files.

Example:

```
for chunk in response.iter_content(chunk_size=1024):
    print(chunk)
```

17. response.iter_lines()

Description: Reads the content as lines, making it useful when dealing with text data.

Example:

```
for line in response.iter_lines():
    print(line)
```

Complete Example

```
import requests
url = "https://jsonplaceholder.typicode.com/posts/1"
response = requests.get(url)
print("Status Code:", response.status_code)
print("Headers:", response.headers)
print("Content:", response.content)
print("Text:", response.text)
print("JSON:", response.json())
print("URL:", response.url)
print("Encoding:", response.encoding)
print("History:", response.history)
print("Cookies:", response.cookies)
print("Elapsed Time:", response.elapsed)
print("Is OK:", response.ok)
print("Apparent Encoding:", response.apparent_encoding)
print("Links:", response.links)
```