

### **Best Practices**

- Catch specific exceptions rather than using a bare except to avoid hiding bugs.
- Use else for code that should only run if no exception was raised.
- Use finally to clean up resources like files or network connections.
- Exceptions can be raised manually using raise.

### **Summary**

Exception handling helps write robust programs that deal with unexpected situations gracefully, improving reliability and user experience. Python's try-except-else-finally blocks provide a flexible framework to catch and respond to runtime errors efficiently without crashing the entire application

## Working with APIs

Working with APIs (Application Programming Interfaces) in Python typically involves sending HTTP requests to retrieve or send data to a web service. Python's most popular library for this purpose is the `requests` module, which is simple and powerful for interacting with RESTful APIs.

### Key Steps to Work with APIs in Python

#### Install the Requests Library

*bash*

```
pip install requests
```

#### Import the Library

*python*

```
import requests
```

#### Making a GET Request

Retrieve data from an API endpoint.

*python*

```
response = requests.get("https://api.example.com/data")
print(response.status_code) # 200 means OK
data = response.json()    # Parse JSON response to Python dict/list
print(data)
```

#### Making a POST Request

Send data to API, e.g., submitting form data.

*python*

```
payload = {"username": "admin", "password": "secret"}
response = requests.post("https://api.example.com/login", data=payload)
print(response.text)
```

#### Handling API Authentication

Many APIs require authentication like API keys or basic auth.

*python*

```
from requests.auth import HTTPBasicAuth
response = requests.get("https://api.example.com/user", auth=HTTPBasicAuth('user', 'pass'))
```

#### Passing Query Parameters

*python*

```
params = {"q": "Python", "page": 1}
response = requests.get("https://api.example.com/search", params=params)
```

#### Handling Errors

Check the response status code or use `response.raise_for_status()` to manage errors.

*python*

```
try:
    response.raise_for_status()
except requests.exceptions.HTTPError as err:
```

```
print(f"HTTP error occurred: {err}")
```

#### Example: Consuming a Public REST API

```
python
import requests

url = "https://jsonplaceholder.typicode.com/posts/1"
response = requests.get(url)
if response.status_code == 200:
    data = response.json()
    print(data)
else:
    print("Failed to retrieve data")
```

#### Summary

- Use the requests library to communicate with APIs through HTTP methods like GET, POST, PUT, DELETE.
- Handle authentication and query parameters as needed.
- Parse JSON responses into Python objects for further use.
- Handle errors gracefully to build robust applications.

Working with APIs in Python opens up vast possibilities for integrating external data and services into applications seamlessly