Requests

HTTP for humans, with Requests' verbs, headers, auth, sessions, and responses at a glance.

HTTP Verbs

One function per HTTP method; each returns a Response.

PURPOSE	COMMAND	RESULT
Read a resource.	`requests.get( "https://api.example.com/items")`	
Create a resource.	`requests.post(url, json={"name": "ada"})`	
Replace a resource.	`requests.put(url, json=data)`	
Partial update.	`requests.patch(url, json={"name": "ada"})`	
Delete a resource.	`requests.delete(url)`	
Headers only / capabilities.	`requests.head(url) .options(url)`	

pick the verb by intent: GET reads, POST creates, PUT/PATCH update, DELETE removes; each returns a Response

Query Parameters

Build the URL query string safely with params.

PURPOSE	COMMAND	RESULT
Add key/value query params.	`requests.get(url, params={"q":"test","page":2})`	
Repeat a key (list value).	`requests.get(url, params={"tag": ["a","b"]})`	
Pass params as a string.	`requests.get(url, params="q=test&page=2")`	
See the final URL.	`r.url`	
Auto URL-encoding of values.	`requests.get(url, params={"q": "a b&c"})`	

params= builds the ?key=value&... query string and URL-encodes special characters; never concatenate URLs by hand

Request Bodies

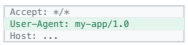
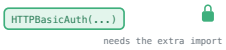
json= for JSON, data= for forms, files= for uploads.

PURPOSE	COMMAND	RESULT
Send JSON (auto-serialize).	`requests.post(url, json={"id": 1, "name": "ada"})`	
Send a form (urlencoded).	`requests.post(url, data={"key": "value"})`	
Upload a file (multipart).	`requests.post(url, files={"file": open(...)})`	
File + extra fields together.	`requests.post(url, data={"caption": "hi"}, files={"file": f})`	
Send raw bytes / text body.	`requests.post(url, data=b"raw bytes")`	

the keyword decides the encoding and Content-Type: json= JSON, data= a form, files= a multipart upload

Headers & Auth

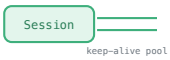
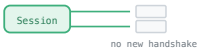
Custom headers, tokens, and built-in auth helpers.

PURPOSE	COMMAND	RESULT
Set custom headers.	`requests.get(url, headers={"User-Agent": ...})`	
Bearer / API-key token.	`requests.get(url, headers={"Authorization": ...})`	
HTTP Basic auth (shortcut).	`requests.get(url, auth=("user", "passwd"))`	
HTTP Basic auth (explicit).	`from requests.auth import HTTPBasicAuth` `auth=HTTPBasicAuth("user", "passwd")`	
Read response headers.	`r.headers["Content-Type"]`	
Headers are case-insensitive.	`r.headers.get("content-type")`	

headers= adds request headers; r.headers is a case-insensitive mapping, so ["Content-Type"] and .get("content-type") match

Sessions

One Session reuses connections, headers, and cookies.

PURPOSE	COMMAND	RESULT
Open a session.	`s =requests.Session()`	
Default headers for every call.	`s.headers.update({"Authorization": ...})`	
Default query params.	`s.params= {"api_key": "abc"}`	
Make requests through it.	`s.get(url) s.post(url, json=data)`	
Cookies persist automatically.	`s.get(login); s.get(profile)`	
Close (or use a context block).	`withrequests.Session() as s: ...`	

a Session reuses the TCP connection (keep-alive) and applies its headers, params, and cookies to every request

Errors & Retries

Check status, catch exceptions, retry transient failures.

PURPOSE	COMMAND	RESULT
Quick success flag.	`ifr.ok: ...`	
Raise on 4xx/5xx.	`r.raise_for_status()`	
Catch any request error.	`exceptrequests.exceptions.RequestException`	
Inspect the status code.	`r.status_coderequests.codes.ok`	
Configure automatic retries.	`Retry(total=3, backoff_factor=0.5, status_forcelist=[...])`	
Mount the retry adapter.	`s.mount("https://", HTTPAdapter(max_retries=...))`	

Requests does NOT raise on 4xx/5xx by default: check r.ok or call raise_for_status(); all errors descend from RequestException

Timeouts & Redirects

Always set a timeout; control how redirects are followed.

PURPOSE	COMMAND	RESULT
Always set a timeout.	`requests.get(url, timeout=10)`	
Separate connect / read timeouts.	`requests.get(url, timeout=(3.05, 27))`	
Timeout raises an exception.	`exceptrequests.exceptions.Timeout`	
Redirects are followed (default).	`r.historyr.url`	
Stop following redirects.	`requests.get(url, allow_redirects=False)`	
Verify TLS / set a CA bundle.	`requests.get(url, verify="ca.pem" | False)`	

without timeout= a request can hang forever; a (connect, read) tuple bounds each phase; keep verify=True so TLS is checked

Reading the Response

Parse JSON, text, or bytes; stream big downloads.

PURPOSE	COMMAND	RESULT
Parse JSON to a dict/list.	`data =r.json()`	
Decoded text (str).	`r.text`	
Raw bytes (downloads).	`r.content`	
Status, reason, timing.	`r.status_coder.reason r.elapsed`	
Stream a large file.	`requests.get(url, stream=True) + iter_content`	
Save bytes to disk.	`open("out.bin", "wb").write(r.content)`	

r.json() parses JSON, r.text gives decoded str, r.content gives raw bytes; stream=True + iter_content keeps memory low