

SI 506 Lecture 20

Topics

1. The Requests package
 1. Install `requests`
 2. Confirm installation
2. The Star Wars API
 1. SWAPI: request resource categories (`/api/`)
 2. SWAPI: request resources by category (`/api/:category/`)
 3. SWAPI: request single resource by id (`/api/:category/:id/`)
 4. SWAPI: search category with querystring (`/api/:category/?search=<search term>`)
 5. SWAPI: search limitations
3. Tips
 1. `requests` module: method chaining
 2. Utility function `get_swapi_resource()`
 3. Other `requests.Response` object properties and methods

Vocabulary

- **API:** Application Programming Interface that specifies a set of permitted interactions between systems.
- **HTTP:** The Hypertext Transport Protocol is an application layer protocol designed to facilitate the distributed transmission of hypermedia. Web data communications largely depends on HTTP.
- **JSON:** Javascript Object Notation, a lightweight data interchange format.
- **Querystring:** That part of a Uniform Resource Locator (URL) that assigns values to specified parameters.
- **Resource:** A named object (e.g., document, image, service, collection of objects) that is both addressable and accessible via an API.
- **URI:** Uniform Resource Identifier that identifies unambiguously a particular resource.
- **URL:** Uniform Resource Locator is a type of URI that specifies the *location* of a resource on a network and provides the means to retrieve it.
- **URN:** Uniform Resource Name is a type of URI that provides a unique identifier for a resource but does not specify its location on a network.

1.0 The Requests package

The authors of the [Requests package](#) describe it as "an elegant and simple HTTP library for Python, built for human beings." With a single line of code you can initiate communication over HTTP with a platform that provides an application programming interface (API) for accessing resources (i.e., objects) remotely. You can utilize the requests package—imported as a module—to create, retrieve, modify, and delete resources stored on servers and accessible via the HTTP protocol.

1.1 Install `requests`

Read the relevant install guide and use `pip` from the command line to install the `requests` package.

- [macOS](#)
- [Windows](#)



If you use [Anaconda](#) see the companion "[Updating the requests package \(Anaconda users\)](#)".

1.2 Confirm installation

Open VS Code or your source code editor/IDE of choice and run the file `swapi_get_test.py` from either the command line (preferred) or directly from inside VS Code or other source code editor/IDE:

```
import requests

# 1.0 SWAPI resource URL (Uniform Resource Locator)
resource = 'https://swapi.py4e.com/api/people/4/'

# 2.0 HTTP GET request / response
# Returns an instance of requests.models.Response
response = requests.get(resource)

# 2.1 response object type
print(f"\nresponse = {type(response)}")

# 3.0 Decode JSON response (to str)
json = response.text # returns a string

print(f"\njson ({type(json)}) = {json}")

# 4.0 Decode JSON response (to dict)
person = response.json() # you will call this regularly

print(f"\nPerson ({type(person)}) = {person}")

# 4.1 Print person name
print(f"\n{person['name']}\n")
```

! if VS Code does not recognize the `import requests` statement in your Python file or you encounter a `ModuleNotFoundError` when running the file after pressing the run button, then VS Code is likely using the wrong Python environment. You can check which Python environment VS Code is using by first checking the Python version listed in the Side Bar (bottom left) of the interface. If it is a version other than what you installed at the beginning of the course, you may need to "repoint" VS Code to the right environment.

To do so, type **Cmd-Shift-P** (macOS) or **Ctrl-Shift-P** (Windows) to activate the Command Palette. Then in the search box type: "Python: Select Interpreter". From the list of options, click on the relevant Python interpreter (e.g., Python 3.9.7 64-bit). You may need to restart VS Code before re-running the file. If the runtime error persists contact the teaching team via Slack.

For more info see, VS Code's ["Using Python Environments in VS Code"](#)

2.0 The Star Wars API (SWAPI)

The [Star Wars API](#) (SWAPI) provides an API for retrieving representations of films, people, planets, species, starships, and vehicles associated with the series situated *a long time ago in a galaxy far, far away*. . . .

! The SWAPI API currently accepts HTTP **GET** requests only (no **PUT**, **POST**, **DELETE** requests accepted).

SWAPI provides the following endpoint for requesting resources:

```
endpoint = 'https://swapi.py4e.com/api'
```

2.1 SWAPI: request resource categories (/api/)

To return a dictionary of key-value pairs that represent the current SWAPI resource categories, add a trailing slash (/) to the endpoint when passing the URL to the `requests.get()` method.

💡 call the response object's `json()` method to "decode" the message as a list or dictionary.

```
response = requests.get(endpoint + '/') # note trailing slash
resources = response.json() # convert message payload to dict

print(f"\n2.1 SWAPI Resources (n={len(resources)})")
for key, val in resources.items():
    print(f"{key.capitalize()}: {val}")
```

💡 For more information on how to utilize SWAPI to retrieve resources, see the SWAPI [documentation](#) page.

! Do *not* use any of the available SWAPI helper libraries to write your code. You will interact with SWAPI using the [requests](#) module and your own code only.

2.2 SWAPI: request resources by category (/api/:category/)

You can retrieve a collection of resources by category (e.g., people) by employing the following URL pattern:

`/api/:category/`

as in

`https://swapi.py4e.com/api/people/`

! Note that SWAPI will *not* respond by returning a collection of all people resources. Instead, SWAPI responses are "paged", with each paged response limited to a max of ten (10) records per request.

```
url = f"{endpoint}/people/"
response = requests.get(url)

payload = response.json() # decode
payload_count = payload['count']
people_returned = len(payload['results']) # SWAPI only returns max 10
records per request
people = payload['results']

for person in people:
    print(person['name'])
```

SWAPI will respond with the following JSON document that provides a paged list of resources stored in a "results" list along with a URI that can be used to retrieve the `next` set of paged resources.

```
{
  "count": 87,
  "next": "https://swapi.py4e.com/api/people/?page=2",
  "previous": null,
  "results": [
    { < Person object > },
    { < Person object > },
    { < Person object > },
    { < Person object > },
    { < Person object > },
    { < Person object > },
    { < Person object > },
    { < Person object > },
    { < Person object > },
    { < Person object > },
    { < Person object > }
  ]
}
```

2.3 SWAPI: request single resource by id (`/api/:category/:id/`)

You can retrieve a single resource by specifying both its category and identifier (number only) by employing the following URL pattern:

`/api/:category/:id/`

as in

`https://swapi.py4e.com/api/people/1/`

```
url = f"{endpoint}/people/1/" # Luke Skywalker
response = requests.get(url) # JSON representation of person returned
person = response.json() # decode to dict
```

2.4 SWAPI: search category with querystring (`/api/:category/?search=<search term>`)

The SWAPI API also facilitates text-based searching of resources. This feature is limited currently to searching on the following attributes:

- `< name >` (all resources)
- `< title >` (Film)
- `< model >` (Starship, Vehicle)

The search feature employs *case-insensitive* partial matches (i.e., contains) on searchable fields.

You can search a category by employing the following URL pattern, which appends a querystring to the URL:

`/api/:category/?search=<search term>`

as in

`https://swapi.py4e.com/api/starships/?search=wing`

Conveniently, the `requests.get()` function defines a second parameter named "params" for passing querystring key-value pairs as dictionary key-value pairs as an optional argument.

```
url = f"{endpoint}/starships/"
params = {'search': 'wing'} # dict
response = requests.get(url, params) # pass search parameters as 2nd
argument

payload = response.json() # decode
starships = payload['results']
```

💡 one can solve the paging challenge by writing a [recursive function](#) that calls itself repeatedly in order to return every paged set of records. But learning how to write such a function is out of scope for SI 506.

2.5 SWAPI: search limitations

1. Only accepts two query string types:
 - a. `search=< string >` (name, title, or model depending on entity)
 - b. `page=< num >`
2. Search employs case-insensitive partial matches on search fields.
3. Response results are paged and limited to a maximum of **10 records** per request.

SWAPI will respond with a JSON document of the requested resource if it exists:

Request

<https://swapi.py4e.com/api/people/?search=luke>

Response

```
{
  "name": "Luke Skywalker",
  "height": "172",
  "mass": "77",
  "hair_color": "blond",
  "skin_color": "fair",
  "eye_color": "blue",
  "birth_year": "19BBY",
  "gender": "male",
  "homeworld": "https://swapi.py4e.com/api/planets/1/",
  "films": [
    "https://swapi.py4e.com/api/films/1/",
    "https://swapi.py4e.com/api/films/2/",
    "https://swapi.py4e.com/api/films/3/",
    "https://swapi.py4e.com/api/films/6/",
    "https://swapi.py4e.com/api/films/7/"
  ],
  "species": [
    "https://swapi.py4e.com/api/species/1/"
  ],
  "vehicles": [
    "https://swapi.py4e.com/api/vehicles/14/",
    "https://swapi.py4e.com/api/vehicles/30/"
  ],
  "starships": [
    "https://swapi.py4e.com/api/starships/12/",
    "https://swapi.py4e.com/api/starships/22/"
  ],
  "created": "2014-12-09T13:50:51.644000Z",
  "edited": "2014-12-20T21:17:56.891000Z",
  "url": "https://swapi.py4e.com/api/people/1/"
}
```

3.0 Tips

3.1 Requests module: method chaining

Note that you can chain the `response.json()` method call to `requests.get()` function call:

```
# Get the Empire Strikes Back (1980)
url = f"{endpoint}/films/"
params = {'search': 'empire strikes back'}
payload = requests.get(url, params).json() # response.json() method
chaining
film = payload['results'][0]


print(f"\n2.5: Film = {film['title']} ({film['release_date']})")
```

Note that you can also add bracket notation to retrieve an element from the returned "results" list:

```
# Get Yoda
url = f"{endpoint}/people/"
params = {'search': 'yoda'}
yoda = requests.get(url, params).json()['results'][0] # whoa
```

3.2 Utility function `get_swapi_resource()`

Given that you will be making frequent requests to the SWAPI endpoint in order to retrieve resources we should implement a function to handle the task. The function `get_swapi_resource` "wraps" the request module's `requests.get()` function and permits sending requests with and without querystring parameters. The function returns a decoded dictionary or list of dictionaries.

 note that the function also sets a timeout value in seconds. This sets a limit on the wait time allowed for a remote service to send back a response. If the wait time exceeds the timeout value an exception is raised. You can test this feature by calling the function and passing to it a timeout value of 0.001.

```
def get_swapi_resource(url, params=None, timeout=10):
    """Returns a response object decoded into a dictionary. If query
    string < params > are
    provided the response object body is returned in the form on an
    "envelope" with the data
    payload of one or more SWAPI entities to be found in ['results'] list;
    otherwise, response
    object body is returned as a single dictionary representation of the
```

SWAPI entity.

Parameters:

url (str): a url that specifies the resource.
 params (dict): optional dictionary of querystring arguments.
 timeout (int): timeout value in seconds

Returns:

dict: dictionary representation of the decoded JSON.
 """

```
if params:
    return requests.get(url, params, timeout=timeout).json()
else:
    return requests.get(url, timeout=timeout).json()
```

3.3 Other `requests.Response` object properties and methods

The return value of the `requests.get()` function is an instance of the `requests.Response` class. The object is provisioned with a number of instance variables and methods. Although you will not be required to make use of these variables or methods you should consider familiarizing yourself with the request module's [API](#) as you learn how to use it.

```
# Get Yoda
url = f'{endpoint}/people/'
params = {'search': 'chewbacca'}
response = requests.get(url, params)

# Status code
print(f"\nResponse status code = {response.status_code}")

# Response headers
print(f"\nResponse headers = {response.headers}")

# Encoding
print(f"\nResponse encoding = {response.encoding}")

# Check for bad request
if response.raise_for_status():
    print(f"\nBad request")
else:
    print(f"\nValid request")

# Decode response
name = response.json()['results'][0]['name']
print(f"\nResource name = {name}")
```