

SI 506 Lecture 17

Topics

1. Install the requests module using pip (pre-req).
2. Understand the basics of the HTTP request/response communication cycle.
3. Learn how to use the requests package to send HTTP GET requests to an endpoint.
4. Learn how to retrieve data from the Star Wars API.
5. Write utility functions to retrieve SWAPI resources and read/write JSON to a file.

Vocabulary

- **API:** Application Programming Interface that species 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.

Previous

- **Argument.** A value passed to a function or method that corresponds to a parameter defined for the function or method.
- **Boolean.** A type (`bool`) or an expression that evaluates to either `True` or `False`.
- **Built-in Function.** A `function` defined by the Standard Library that is always available for use.
- **Caller.** The initiator of a function call.
- **Conditional Statement.** A statement that determines a computer program's *control flow* or the order in which particular computations are to be executed.
- **Deep copying.** For a given mutable object (e.g., `list`) constructs a new compound object and recursively *copies* into it objects found in the original.
- **Dictionary.** An associative array or a map, wherein each specified value is associated with or mapped to a defined key that is used to access the value.
- **Expression.** An accumulation of values, operators, and/or function calls that return a value. `len(<some_list >)` is considered an expression.
- **f-string.** Formatted string literal prefixed with `f` or `F`.
- **File Object.** An object that provides a file-oriented application programming interface (API) to a either a text file, binary file (e.g., image file), or a buffered binary file. File objects include read and write methods for interacting with a file stored locally or remotely.

- **Flow of execution.** The order in which statements in a program are executed. Also referred to as *control flow*.
- **Function.** A defined block of code that performs (ideally) a single task. Functions only run when they are explicitly called. A function can be defined with one or more *parameters* that allow it to accept *arguments* from the caller in order to perform a computation. A function can also be designed to return a computed value. Functions are considered "first-class" objects in the Python eco-system.
- **Immutable.** Object state cannot be modified following creation. Strings are immutable.
- **Iterable.** An object capable of returning its members one at a time. Both strings and lists are examples of an iterable.
- **Iteration.** Repetition of a computational procedure in order to generate a possible sequence of outcomes. Iterating over a `list` using a `for` loop is an example of iteration.
- **Method.** A function defined by and bound to an object. For example the `str` type is provisioned with a number of methods including `str.strip()`.
- **Mutable.** Object state can be modified following creation. Lists are mutable.
- **Nested Loop.** A `for` or `while` loop located within the code block of another loop.
- **Operator.** A *symbol* for performing operations on values and variables. The assignment operator (`=`) and arithmetic operators (`+`, `-`, `*`, `/`, `**`, `%`, `//`).
- **Parameter.** A named entity in a function or method definition that specifies an argument that the function or method accepts.
- **Scope.** The part of a script or program in which a variable and the object to which it is assigned is visible and accessible.
- **Sequence.** An ordered set such as `str`, `list`, or `tuple`, the members of which (e.g., characters, elements, items) can be accessed.
- **Shallow copying.** For a given mutable object (e.g., `list`) constructs a new compound object but inserts *references* (rather than copies) into it of objects found in the original. The `list.copy()` returns a shallow copy of the original list.
- **Slice.** A subset of a sequence. A slice is created using the subscript notation `[]` with colons separating numbers when several are given, such as in `variable_name[1:3:5]`. The bracket notation uses slice objects internally.
- **Statement.** An instruction that the Python Interpreter can execute. For example, assigning a variable to a value such as `name = 'arwhyte'` is considered a statement.
- **Truth Value.** In Python any object can be tested for its *truth value* using an `if` or `while` condition or when it is used as an operand in a *Boolean operation*.
- **Tuple.** An ordered sequence that cannot be modified once it is created.
- **Tuple packing.** Assigning items to a tuple.
- **Tuple unpacking.** Assigning tuple items to an equal number of variables in a single assignment. A `list` can also be unpacked.

1.0 The requests package

The authors of the [requests package](#) describes 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 system 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](#)

1.2 Confirm installation

Open VS Code or your source code editor/IDE of choice and run the file `swapi_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 green 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 3.9.2 64-bit). You may need to restart VS Code before re-running the file. If the runtime error persists contact the teaching team via Piazza.

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 **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 Get SWAPI 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 with a paged list of resources stored in a "results" list along with a URL 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 > }
  ]
}

```

2.3 SWAPI: request single instance 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

print(f"\n2.3: Request person by id = {person}")

```

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 pagged and limited to 10 records max per request.

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

```
{
  "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/"
}
```

2.4 SWAPI: search category by resources (`/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: 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

print(f"\n2.5: Yoda = {yoda}")
```

2.6 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.

```
# 2.6 Additional Response properties
# 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}")
```

3.0 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()
```

4.0 json module

Like the `csv` module the Python standard library's `json` module provides enhanced functionality for working with JSON files. JSON is a lightweight data interchange format for exchanging information between systems.

To use the `json` module you must import it in your Python file:

```
import json
```

There are four `json` module functions; two of which are of particular interest to us:

1. `json.load()` -- *deserializes* (decodes) a text or binary file that contains a JSON document to a `dict` or `list`.
2. `json.loads()` -- *deserializes* (decodes) a string, bytes, or bytearray containing a JSON document to a `dict` or `list`.
3. `json.dump()` -- *serializes* (encodes) an object as a JSON formatted stream to be stored in a file.
4. `json.dumps()` -- *serializes* (encodes) an object to a JSON formatted string.

Since you will also be working with JSON documents between now and the end of the semester implementing a function that can read a JSON document as well one that can write a JSON document to a file will prove useful.

4.1 Reading JSON files (`json.load()`)

The function `read_json()` reads a JSON document per the provided filepath, calls the `json` module's `json.load()` function in order to *decode* the file data as a `dict` or a `list` (of dictionaries), and returns the decoded data to the caller.

```
def read_json(filepath, encoding='utf-8'):
    """Reads a JSON document, decodes the file content, and returns a list
    or
    dictionary if provided with a valid filepath.

    Parameters:
        filepath (str): path to file
        encoding (str): name of encoding used to decode the file

    Returns:
        dict/list: dict or list representations of the decoded JSON
    document
    """

    with open(filepath, 'r', encoding=encoding) as file_obj:
        return json.load(file_obj)
```

4.2 Writing to a JSON file (`json.dump()`)

The function `write_json()` accepts a dictionary or a list of dictionaries, calls the `json` module's `json.dump()` function in order to *encode* the passed in data as JSON, and writes the encoded data to the target file.

```
def write_json(filepath, data, encoding='utf-8', ensure_ascii=False,
indent=2):
    """Serializes object as JSON. Writes content to the provided filepath.

    Parameters:
        filepath (str): the path to the file
        data (dict)/(list): the data to be encoded as JSON and written to
    the file
        encoding (str): name of encoding used to encode the file
        ensure_ascii (str): if False non-ASCII characters are printed as
    is; otherwise
                            non-ASCII characters are escaped.
        indent (int): number of "pretty printed" indentation spaces applied
    to encoded JSON

    Returns:
        None
    """

    with open(filepath, 'w', encoding=encoding) as file_obj:
        json.dump(data, file_obj, ensure_ascii=ensure_ascii,
        indent=indent)
```