Contents

1 week 6: apis

Agenda

- look at LOC standards for XML
- begin work with APIs
  - dictionary structures - new data structure
  - the MET API - beginner friendly
  - end goal: to get data from MET databases into our own spreadsheet, curate that data a little bit.

1.1 HTML and XML

Look at standards from LOC.

1.2 the dictionary data type

What is a dict? key:value pairs

```
  filipa = {
    'name': ['filipa', 'da gama', 'calado'],
    'age': 35,
    'degree': 'literature',
    'jobs': ['professor', 'digital scholarship specialist', 'student']

  # access items with brackets, the 'key'

  filipa['jobs'] # 'professor'...
  filipa['age'] # 35

  # access items within a column, like list indexing

  filipa['name'][1] # 'da gama'
}
```

1.3 the MET API

Order of things we will do

- construct the API request (URL)
- make the request
- parse/format our response
- with information from that response, make another call to get more information.

1.3.1 the anatomy of an API request:

3 parts of an API: the base url, the path, the query.

```
base_url = "https://collectionapi.metmuseum.org" # root of the website
path = "/public/collection/v1/search" # directory structure, folders
query = "?q=woman" # parameter, endpoint - keyword/specifications

# our request
women = requests.get(f'{base_url}{path}{query}')
```

```
# should say 200
women
```

All you need for an API is a URL. An HTTP request.

Some URLs contain requests embedded inside of them. If you know how to look.

1.3.2 inspecting objects: two functions: type() and dir()

Using type() and dir() to better understand our response data. The end goal is to sift through the data to discover interesting things.

```
# what type of object do we have?

type(women)

# What can we do with a Response object? Spend a couple of minutes exploring the diffe
dir(women)
```

1.3.3 .json() to parse our response object

```
# why do we need to add parenthesis?
  parsed = women.json()

  # what is it
  type(parsed)

  # what can we do with it
  dir(parsed)

  # check out the keys
  parsed.keys()
```

1.3.4 accessing items from a dict

```
# how would we get the object ids?

parsed['objectIDs']

# how would we get just the first object id?

first = parsed['objectIDs'][0]
```

1.3.5 using the "objects" endpoint

We can use this first object to create a new request. This time, using a different path, the "objects" path.

```
objects_path = '/public/collection/v1/objects/'

url = f'{base_url}{objects_path}{first}'

first_object = requests.get(url)
```

Inspecting our new response object, parsing the results

```
# checking the resulting object
first_object

# the response is just like before.
type(first_object)
dir(first_object)

# parsing the object:

first_object.json()
```

See all the data about this object.
See the objectURL? Copy and paste it into your browser.

1.3.6 reading documentation: individual challenge

Look at the docs, see the different paths and parameters.

Pratice searching using another parameter for the search endpoint. And then, take the results of that search and look for the objects.

1.3.7 looping through our objects

We are going to automate the collection of object metadata. How?

Making a list of objectIDs that we can loop through, and for each one, make a new request using the "objects" path.

```
# just the first ten
ids = parsed['objectIDs'][:10]

# start small, just print the ids for now
for item in ids:
  print(item)

# add pseudocode, then fill it in
# what are the steps we need to take?

first_ten = []
for item in ids:

    # passing the objectID variable into the URL
    base_url = 'https://collectionapi.metmuseum.org'
    path = '/public/collection/v1/objects/'
    url = f'{base_url}{path}{item}'

    # grabbing our response for that object
    response = requests.get(url)

    # parsing our response with json
    parsed = response.json()

    # appending the response to our new list
    first_ten.append(parsed)

# check the content.
type(first_ten)
```

Write a loop to print out information about some of the keys for each object. For example, print out the value of artistDisplayName.

```
for item in first_ten:
  print(item['artistDisplayName'])
```

1.3.8 group challenge

Google the error.

There are two ways: conditional statement and creating a variable

```
for item in first_ten:
  if item.get('artistDisplayName'):
      print(item['artistDisplayName'])
```

```
for item in first_ten:
    title = item.get('artistDisplayName')
    print(title)
```

1.3.9 about "women"

Try it with gender. Why do you think we see only female?

1.3.10 saving to DataFrame

1.3.11 data anlaysis with pandas