nustem

Electricity Generation API

Query the National Grid for current supply fuel mix, then draw a graph.

Ingredients

Platform:	Raspberry Pi / Raspbian Python3
Components:	None.
Libraries:	requests matplotlib

Glossary

JSON	"JavaScript Object Notation" – a standardised way of packaging structured data so the structure can be unpacked by a program.
parsing	We parse a sentence by breaking it into its constituent parts. Similarly, 'parsing' in computing means 'to break down into logical components.'

Fetching data from the web

There are thousands of data feeds available over the web, both public and private. Typically, you load a documented web address and receive structured data in response. Because the data's structured, you can write programs which *parse* it – break it down into meaningful chunks.

In this example, we'll query the National Grid's web interface for information about the current fuel mix for electricity generation, and draw a neat pie chart of the response.

We'll use Python's `requests` library to retrieve *JSON-formatted data*. We'll parse that using Python's handy *iterators*, storing the data in *lists*, which we'll then pass directly to a graph-drawing library called *MatPlotLib*. This is a very real-world sort of example: we're retrieving data, processing it via an intermediate representation, then handing it off to a third-party module which does the hard work for us.

elecgenapi-1.py

Open this file in Thonny; it's in `Digital-Making/examples/elecgenapi/`. Run the code, see what it prints out, then we'll step through it:

The program uses the `requests` library to retrieve structured data from a web service provided by the National Grid. This line:

```
r = requests.get('https://api.carbonintensity.org.uk/generation')
```

...sets up an object `r` which will hold the response `requests` gets, in this case a report of the fuel sources for the last 30 minutes of UK energy generation. `r` can be asked to provide that data in a range of ways. We ask it for JSON:

```
mix = r.json
```

If you were to `print(mix)` at this point you'd see a mess of brackets like this:

```
{'data': {'from': '2019-08-07T09:30Z', 'to': '2019-08-07T10:00Z',
'generationmix': [{'fuel': 'biomass', 'perc': 6.9}, {'fuel':
'coal', 'perc': 2.7}, {'fuel': 'imports', 'perc': 4.9}, {'fuel':
'gas', 'perc': 30.4}, {'fuel': 'nuclear', 'perc': 18.2}, {'fuel':
'other', 'perc': 0.3}, {'fuel': 'hydro', 'perc': 2.4}, {'fuel':
'solar', 'perc': 12.5}, {'fuel': 'wind', 'perc': 21.7}]}}
```

Northumbria University NEWCASTLE

Hey, at least it's text. You're looking at JSON.

JSON data

JavaScript Object Notation is a standardised way of wrapping up structured data so it's readable by computers and sort-of readable by humans. Click the 'Load' button in Thonny, then ask it to show you 'all files' rather than just Python files, and load `example.json`:

```
{
  "data": {
    ...
    "generationmix": [
      { "fuel": "biomass", "perc": 7.1 },
      { "fuel": "coal", "perc": 0 },
      ...
      { "fuel": "wind", "perc": 2.8 }
    ]
  }
}
```

This is just the previous data reformatted. If you squint between the brackets, this is starting to make sense. Assuming '`perc`' means 'percentage', you could even draw a graph. To do that, you'd extract the name of each fuel and its corresponding percentage value. Let's do that in Python:

Parsing JSON

Neither Joe nor I understood the original paragraph I'd written here. For this exercise, just roll with `mix['data']['generationmix']` being the lines of the data structure above which contain the `fuel` and `perc` data. So if we step through this line-by-line we can extract each of the fuel types and percentages in turn.

(Read up on Python dictionary keys if you want a bit more information on how this works, but there are enough ideas flying around in this worksheet, so do that later!)

Python iterators

One of Python's great strengths is that any time you have a list of similar things like this, you can step through them by naming a temporary variable which holds each line in turn. You can call that temporary variable anything, but in this case it makes sense to call it `fuel`:

```
for fuel in mix['data']['generationmix']:
    fueltype = fuel['fuel']
    percentage = fuel['perc']
    print(fueltype + ": " + str(percentage))
```

For each line of the JSON data, this prints out the fuel type and corresponding percentage, automatically exiting after the last line of data.

Note that we're only storing the fuel type and percentage temporarily. We'll need to fix that in a moment.

There's one other tricky bit: `str(percentage)`. Try removing the `str(…)` wrapper and see what happens. Can you explain?

Storing data in lists

Open `elecgenapi-2.py` in Thonny. You'll notice a few minor changes in the way we're calling `requests.get`, and near the top of the file:

```
import matplotlib.pyplot as plt
```

This brings in some useful bits of the Matplotlib library, which is going to handle drawing a graph for us.

Our immediate challenge, however, is to store the fuel type and percentage data rather than just print it out. We're going to make two empty lists:

```
fueltype = []
percentage = []
```

...then we'll modify the iterator to add each line's data to the end of the corresponding list:

```
for fuel in mix['data']['generationmix']:
    fueltype = fuel['fuel']
    percentage = fuel['perc']
    fueltype.append(fueltype)
    percentage.append(percentage)
```

In `elecgenapi-2.py` you'll notice that I've simplified this a little, by no longer storing each value in the intermediate variables fueltype and percentage. The resulting code is shorter, but the brackets are a bit more confusing. Neither way is 'correct'.

At this point we have two lists, `fueltype` and `percentage`. Let's chuck them at Matplotlib and get it to draw a chart for us.

Matplotlib

Matplolib is a sprawling library for drawing graphs and charts, and you could spend weeks exploring it. Luckily, there's a pie chart example here:

```
https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html
```

...which does very nearly what we want. First we ask matplotlib for a new figure, containing chart axes:

```
fig1, ax1 = plt.subplots()
```

...then we throw data at those axes, telling it we're drawing a pie chart specifically. The data we're plotting is in `percentage`, with each segment label in `fueltype`. The '`autopct`' bit formats the percentages for cleaner display.

```
ax1.pie(percentage, labels=fueltype, autopct='%1.1f%%', [...] )
```

Now we tell the chart the axes are equal so it draws a round pie.

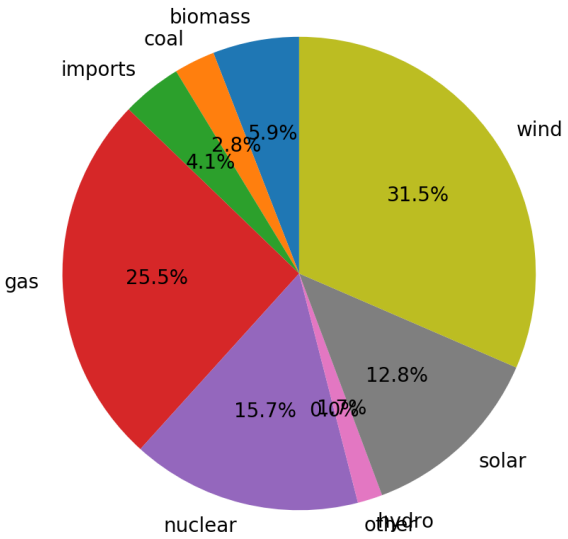
```
ax1.axis('equal')
```

Finally, we ask matplotlib to display everything it knows about:

```
plt.show()
```

Confession: I don't really understand matplotlib. But that doesn't stop me from using it to draw graphs when I need to.

Run `elecgenapi-2.py`, and (after a while, if this is the first run) matplotlib will display a window with a nice pie chart in it.



Well, a fairly nice pie chart.

Exercise: Draw a bar chart

It's easier to draw a bar chart than a pie chart.

Delete all the pie chart gubbins:

```
fig1, ax1 = plt.subplots()
ax1.pie(percentage, labels=fueltype, autopct='%1.1f%%', [...] )
ax1.axis('equal')
```

...then access the `plt` object directly:

```
plt.bar(fueltype, percentage)
plt.show()
```

You can add labels with lines like:

```
plt.ylabel('Percent')
plt.title('Electricity generation by fuel source')
```

You'll need to add these lines before telling the plot to show itself.

See if you can get a nice bar chart to display.

Can you label it with the time the data was collected?