10 Reading data in standard formats

Until now, we have always read data from files, and these have almost always been tabular - data in rows and values in columns, separated by commas or -- exceptionally and usually just for practice -- something else.

Not always so. Data can also be written in other formats. Sometimes for no particular reason, but often because the data will be structured: in csv we can only write tables -- rows and columns -- but not more complex, say hierarchical data. A contact list is a list of people, each person can contain several addresses, phone numbers and email addresses, all of them further characterised by whether they are work, home or other, various notes and attachments can be added... It would be cumbersome to record such data in tabular form. We therefore use other formats. Fortunately, there are only a few and Python is well equipped to read them all.

1. **Reading data from the web**

As we will often get our data online, not in ready-made files, let's start by learning how to read data from a web page. More specifically, from a web resource that is not really a web page for humans to read, but for computers to take data from. To get started, we will read the NLB course book, which can be obtained from https://www.nlb.si/services/tecajnica/?type=companies&format=csv.

Let's copy the URL into a browser, have a look.

```

Num Date\_\_\_\_ Bank Type\_\_\_\_\_\_\_ NCu CCu Buy\_\_\_\_\_\_\_\_\_\_\_ Sell\_\_\_\_\_\_\_\_\_\_

001 20231227 NLB\_ companies\_\_ 840 USD 0001,111300000 0001,095000000

001 20231227 NLB\_ companies\_\_ 826 GBP 0000,873800000 0000,860800000

001 20231227 NLB\_ companies\_\_ 756 CHF 0000,951700000 0000,933700000

001 20231227 NLB\_ companies\_\_ 348 HUF 0383,700000000 0379,700000000

001 20231227 NLB\_ companies\_\_ 941 RSD 0119,470000000 0114,870000000

001 20231227 NLB\_ companies\_\_ 977 BAM 0001,968000000 0001,947000000

001 20231227 NLB\_ companies\_\_ 807 MKD 0062,300000000 0060,500000000

001 20231227 NLB\_ companies\_\_ 203 CZK 0024,760000000 0024,420000000

001 20231227 NLB\_ companies\_\_ 985 PLN 0004,354000000 0004,314000000

001 20231227 NLB\_ companies\_\_ 975 BGN 0001,967000000 0001,937000000

001 20231227 NLB\_ companies\_\_ 578 NOK 0011,292000000 0011,152000000

001 20231227 NLB\_ companies\_\_ 752 SEK 0011,112000000 0010,972000000

001 20231227 NLB\_ companies\_\_ 208 DKK 0007,504900000 0007,404900000

001 20231227 NLB\_ companies\_\_ 036 AUD 0001,624200000 0001,610200000

001 20231227 NLB\_ companies\_\_ 124 CAD 0001,463400000 0001,449400000

001 20231227 NLB\_ companies\_\_ 392 JPY 0158,140000000 0156,740000000

001 20231227 NLB\_ companies\_\_ 946 RON 0005,009200000 0004,929200000

001 20231227 NLB\_ companies\_\_ 710 ZAR 0020,701400000 0020,301400000

001 20231227 NLB\_ companies\_\_ 344 HKD 0008,724100000 0008,524100000

001 20231227 NLB\_ companies\_\_ 949 TRY 0032,628500000 0032,208500000

001 20231227 NLB\_ companies\_\_ 484 MXN 0019,047700000 0018,447700000

001 20231227 NLB\_ companies\_\_ 554 NZD 0001,760100000 0001,728100000

001 20231227 NLB\_ companies\_\_ 376 ILS 0004,041500000 0003,951500000

001 20231227 NLB\_ companies\_\_ 784 AED 0004,076000000 0004,036000000

```

I chose this site precisely because it gives us a course book in a format we already know. The more "official" Bank of Slovenia exchange rate book is in XML, which we still need to learn to read.

(You may notice, by the way, that the URL ends with `format=csv`. Obviously it could say something else, say `format=xml`. To learn more about how you can change the URL to get different information, see https://www.nlb.si/avtomatiziran-prevzem-tecajnic. NLB documents this; we will often have to guess the URL format ourselves or use appropriate tools to discover it.)

Now our job is not to get this data into a browser, but to read it into a Python program. We need to use the `urllib.request` module; this contains the `urlopen` function, which behaves a bit like the `open` function we use to open files: just as `open` returns an object representing a ready-to-read file, so `urlopen` returns something that is ready-to-read.

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Now, what kind of reader from `csv`?

If you open a file with `open` (and don't ask it to open as a non-text, binary file), you will get strings when you read the lines (or the whole file). The `urlopen` function returns an object that returns bytes instead of strings when it is read.

Ein Bild, das Text, Screenshot, Schrift, weiß enthält.

Automatisch generierte Beschreibung

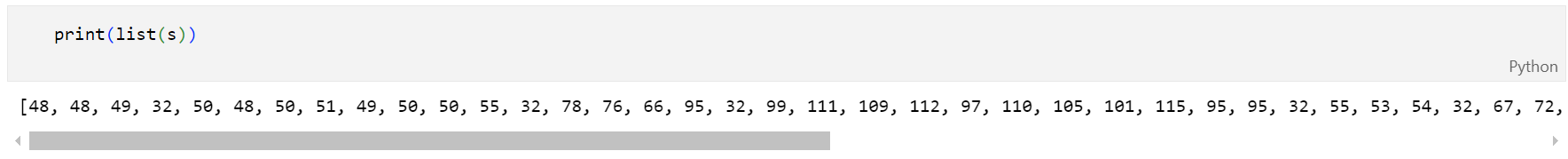
Do we see `b` at the beginning, before the quotation marks? This tells us that what is between the quotation marks is not a sequence of characters (a string) but a sequence of numbers. It shows the numbers that can be shown as ASCII characters as characters, and replaces the rest with something strange (which we thankfully don't see here).

We can see the difference between strings and bytes by trying to access individual elements.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

If `s` were a string, we would get the 14th character "N". But this gives the fourteenth digit, 76 (which happens to be a capital N in ASCII). We also see the difference when converting to a list.



If `s` were a string, you would get a list of letters, but this is a list of numbers.

Why? Both files and web data are just numbers. When reading files, Python assumes that the numbers represent characters written in the default encoding for the operating system (UTF-8 on decent systems (Linux, macOS), various local encodings on non-decent ones), or the encoding passed to the `open` function as an additional `encoding` argument. It therefore \*decodes\* all the numbers it reads into strings.

When reading from the web, Python (or the object returned by `urlopen`) does not decode anything. It just returns the numbers, and we have to decode them by calling the `decode` method ourselves.



This one was decoded in ASCII, as there are no alphabets in currency names (at least in this list). If we were reading some other data, we might write

Ein Bild, das Text, Screenshot, Schrift, weiß enthält.

Automatisch generierte Beschreibung

Or even:

Ein Bild, das Text, Screenshot, Schrift, weiß enthält.

Automatisch generierte Beschreibung

Whatever.

We'll just read the course book without the `csv`, because it won't be that complicated.

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

Automatisch generierte Beschreibung

Since NLB publishes machine-readable data by using a decimal point instead of a period, we need to use `replace(",", ".")` to put the data into machine-readable form before calling `float`.

1. **JSON**

One of the common formats in which we get data from the web is JSON. Because web pages are usually programmed in JavaScript, they prefer to send data in JavaScript's own format

To read and write JSON, Python has a `json` module:

- `json.dump(obj, f)` dumps the value of `obj` in json format into the file `f`,

- `json.load(f)` reads the (next) object from `f`,

- `json.dumps(obj)` returns a string containing the object `obj` in json format, and,

- `json.loads(s)` parses the object from the string s.

Let's see what our hinge would look like in it.

Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

Yes. If you write a Python dictionary in json format, it looks exactly like a Python dictionary. :)

And the list?

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

In short. If we store Python objects in json format, we write them exactly as we write them in Python. Almost.

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

It writes them in Javascript: `True` and `False` are written in lowercase and `null` is written instead of `None`.

Terks become lists.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

He doesn't know about the masses.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

Tuples and sets won't hurt us much: in Python, we don't usually store things in JSON (except when we're doing it to send data somewhere, and the recipient will typically be JavaScript, which doesn't like sets and terks anyway). We'll read the data in JSON, and it'll be whatever they put in there. We'll be able to read everything.

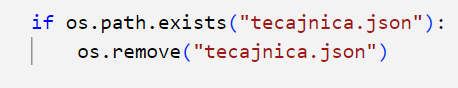
Nevertheless: let's write the courseware in JSON. Let's also add a suitable motivation: if we were to write a currency conversion program, it would read the data from the NLB every time we run it. It might be more convenient to store it in a file. The program would then work by retrieving the data from the web only the first time, and then reading it straight from the file.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Now let us consider that the hinge actually changes daily. When we read the file, we need to check that it contains today's data. The easiest way to do this is to write the date in it.

(To avoid the file we have just written getting in the way in the next step, let's delete it here.)



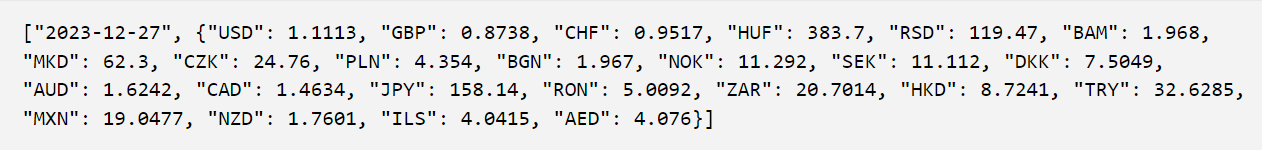
Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

„Tečajnico berem iz datoteke.“

The first time the top cell is run, it says it is reading from the web. On subsequent runs, it says it is reading from a file. If we wait a day (or run one cell higher, the one that deletes the file), it will read from the web again.

The file tecajnica.json looks like this.



1. **Pickle**

Pickle is not a standard format. You won't find data written in it on the web. But it is useful if you want to save any Python object (or several objects) to a file in a quick and easy way.

We use it in a similar way to json.

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Pickle does not write to a text file, so it is not offered a file opened with \*\*`open(name)` but rather `open(name, "wb")` or when reading `open(name, "rb")`\*\*. The second letter, "b", indicates that it is a binary file.

Pickle can write anything. (Say. It is possible to define a new data type that Pickle doesn't understand, but it takes almost no effort.) The problem with Pickle is that only Python knows this notation format and, worse, the notation changes and a newer version of Python can compose a new format that is not readable in older ones.

If we want to save something to read later - by ourselves, to ourselves - pickle is the simplest.

1. **XML**

JSON is designed to carry data. However, data that is made publicly available to be read by computers will most often be in XML format.

For those who have ever seen HTML, XML, the \*Extensible markup language\*, will look familiar. Let's start with hand-crafted XML that could be used to describe contact information. In the notes, we find them in the file clovek.xml.

Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

Automatisch generierte Beschreibung

As we can see, XML consists of a hierarchy of elements. A file always has one basic, root element - here it is `person`. This is "opened" at the beginning, with `<person>` and closed at the end, with `</person>`. In between are other elements, say `name` (starting with `<name>` and ending with `</name>`), and `last name`, `age` and so on. Within the `contacts` element, there are, if we so agree, elements of the type `phone` and `mail`, each of which may be repeated several times.

The `phone` element may have the `type` attribute, and the `mail` element may have the `official` attribute.

We can see how to format a thing by an example. There is a lot more that could be said about XML, but here, for this subject, let us restrict ourselves to the basic idea. The details will be found elsewhere by those who need them.

XML reading libraries work in two ways - this is true not only for Python but also in general. The first type works by reading the file and informing us of the elements it encounters. This way of reading takes up less memory, but is more complex to use. Here we will look at the simpler one, which works by reading the whole file into a tree structure and then allowing us to search through it.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

For reasons that will be explained later, instead of clovek.xml we read clovek-compressed.xml, which is identical to clovek.xml, except that we compressed everything into one line and deleted the spaces between the elements.

The `contact` variable now contains the document being read. It has a bunch of properties and methods, but we'll only be interested in one: `getElementsByTagName`. This is how, say, we get all the elements of `phone`:

Ein Bild, das Text, Screenshot, Schrift, weiß enthält.

Automatisch generierte Beschreibung

Let's play with the first of them.



The `el` element has brothers. The first is `nextSibling`.

Ein Bild, das Text, Screenshot, Schrift, weiß enthält.

Automatisch generierte Beschreibung

She also has a brother.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

The brother of the first phone is the second phone. The brother of the second phone is the first mail. And so on.

An item also has a parent.

Ein Bild, das Text, Screenshot, Schrift, weiß enthält.

Automatisch generierte Beschreibung

The father of the telephone is Edison. The father of the `phone' element is `contacts'.

The father has children - these are the "children" of the `contacts' element - telephones and mails.

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

The `getElementsByTagName` method does not have only the document but each element. So we can get the mails inside the `element`.

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

The element also has children. Actually, only one, so we won't go through the `childNodes`, we'll just go with `firstChild`.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

The element that is the child of `phone` is `Text Node`. This has no children, but has a value, `nodeValue`.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

Oh, and the attributes.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

That's it. Let's summarise everything we will need.

- `el.getElementsByTagName(name)` returns the elements named `name` within `el`.

- `el.getAttribute(name)` returns the value of the attribute `name`.

- The `el.childNodes` are the children of the element.

- The `el.firstChild` is the first child.

- The `el.nextSibling` is the next sibling.

- The `el.parentNode` is the father of the element.

- `el.nodeValue` is the value within the element.

Now we know how to read most of the XML we will come across.

1. **Why compressed-clover.xml?**

If an XML file is formatted so that it can be read by mortals, not just computers, it contains spaces and newline characters. These are seen as additional elements of the Text Node type. In principle they wouldn't (and won't) bother us, only the `nextSibling` of a phone or email would be a Text Node, and they would also appear in all the children's lists, spoiling the simplicity of the example. Just this.

1. **Bank of Slovenia exchange rates**

The Bank of Slovenia publishes the latest exchange rates in XML format at <https://www.bsi.si/_data/tecajnice/dtecbs.xml>.

```xml

<DtecBS xmlns="http://www.bsi.si" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.bsi.si http://www.bsi.si/\_data/tecajnice/DTecBS.xsd">

<tecajnica datum="2023-12-22">

<tecaj oznaka="USD" sifra="840">1.1023</tecaj>

<tecaj oznaka="JPY" sifra="392">156.66</tecaj>

<tecaj oznaka="BGN" sifra="975">1.9558</tecaj>

<tecaj oznaka="CZK" sifra="203">24.589</tecaj>

<tecaj oznaka="DKK" sifra="208">7.4560</tecaj>

<tecaj oznaka="GBP" sifra="826">0.86660</tecaj>

<tecaj oznaka="HUF" sifra="348">381.93</tecaj>

<tecaj oznaka="PLN" sifra="985">4.3420</tecaj>

<tecaj oznaka="RON" sifra="946">4.9708</tecaj>

<tecaj oznaka="SEK" sifra="752">11.0556</tecaj>

<tecaj oznaka="ISK" sifra="352">150.50</tecaj>

<tecaj oznaka="CHF" sifra="756">0.9417</tecaj>

<tecaj oznaka="NOK" sifra="578">11.2705</tecaj>

<tecaj oznaka="TRY" sifra="949">32.2044</tecaj>

<tecaj oznaka="AUD" sifra="036">1.6197</tecaj>

<tecaj oznaka="BRL" sifra="986">5.3624</tecaj>

<tecaj oznaka="CAD" sifra="124">1.4639</tecaj>

<tecaj oznaka="CNY" sifra="156">7.8640</tecaj>

<tecaj oznaka="HKD" sifra="344">8.6105</tecaj>

<tecaj oznaka="IDR" sifra="360">17029.65</tecaj>

<tecaj oznaka="ILS" sifra="376">3.9764</tecaj>

<tecaj oznaka="INR" sifra="356">91.6280</tecaj>

<tecaj oznaka="KRW" sifra="410">1430.05</tecaj>

<tecaj oznaka="MXN" sifra="484">18.6955</tecaj>

<tecaj oznaka="MYR" sifra="458">5.1059</tecaj>

<tecaj oznaka="NZD" sifra="554">1.7505</tecaj>

<tecaj oznaka="PHP" sifra="608">61.067</tecaj>

<tecaj oznaka="SGD" sifra="702">1.4593</tecaj>

<tecaj oznaka="THB" sifra="764">38.084</tecaj>

<tecaj oznaka="ZAR" sifra="710">20.3070</tecaj>

</tecajnica>

</DtecBS>

* The structure is obvious. Let's read it!



Quite simple, isn't it?

At https://www.bsi.si/\_data/tecajnice/dtecbs-l.xml we can find the courses from 1 January 2007. Since this XML takes a long time to download, we already have it saved alongside the notes, so we'll just read it from the file.

The file is structured as follows.

```xml

<?xml version="1.0"?><DtecBS xmlns="http://www.bsi.si" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.bsi.si http://www.bsi.si/\_data/tecajnice/DTecBS-l.xsd">

<tecSBSBS date="2007-01-01">

<tecaj code="USD" sifra="840">1.3170</tecaj>

<here code="JPY" code="392">156.93</here>

<tecaj code="BGN" code="975">1.9558</tecaj>

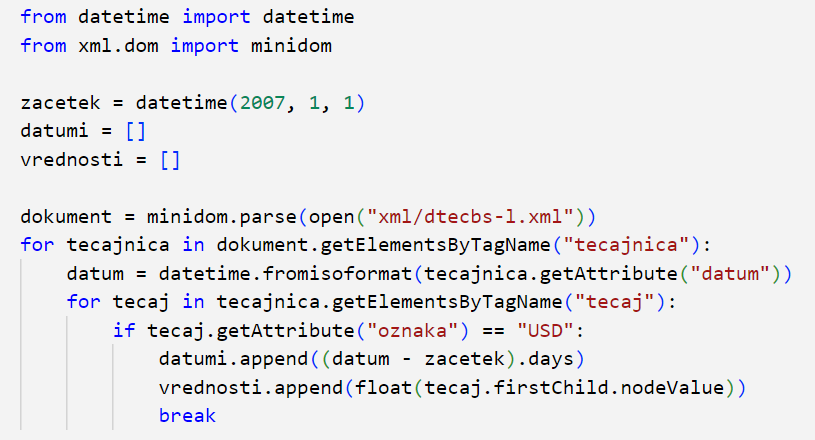
...

```

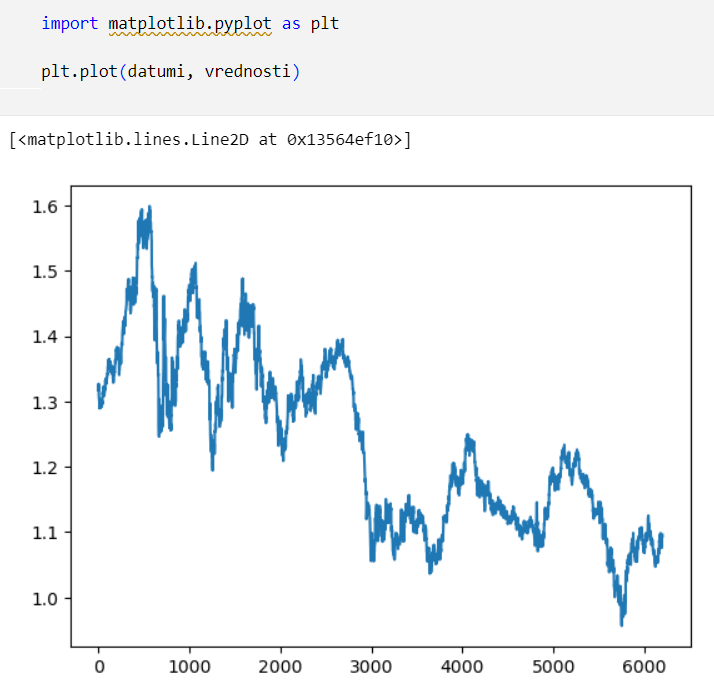
Same as the previous one, except that the rates for each day are also enclosed in `tecajnica`, which contains the date as an attribute.

Let's read, say, the dollar rate. We will read in two lists: one will contain the dates, the other the rate. The dates will not be written in days, years and so on, but in the number of days that have passed since 1 January 2007. For this we will use `datetime`; this has a method `fromisoformat` that can read the date as it is written in this file. We will subtract 1 January 2007 from the date; the result will be some `timeday` data type that has `days` attributes. This contains the difference in days.

Let's go!



And how is he doing, dollar? Like this.



1. **XML is everywhere**

If you can read XML, you can read almost anything.

Have you ever planned a route and got a .gpx file? Or done a route and saved it on Strava or something? You can export that to GPX too. GPX is XML in some standardised format.

This is how Strava saves your run. It contains time location and altitude information, plus whatever your watch (or whatever you use to feed your ego during and after your run) is recording.

```xml

<?xml version="1.0" encoding="UTF-8"?>

<gpx creator="StravaGPX" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd http://www.garmin.com/xmlschemas/GpxExtensions/v3 http://www.garmin.com/xmlschemas/GpxExtensionsv3.xsd http://www.garmin.com/xmlschemas/TrackPointExtension/v1 http://www.garmin.com/xmlschemas/TrackPointExtensionv1.xsd" version="1.1" xmlns="http://www.topografix.com/GPX/1/1" xmlns:gpxtpx="http://www.garmin.com/xmlschemas/TrackPointExtension/v1" xmlns:gpxx="http://www.garmin.com/xmlschemas/GpxExtensions/v3">

<metadata>

<time>2023-12-16T14:22:38Z</time>

</metadata>

<trk>

<name>Dragomelj z repom</name>

<type>running</type>

<trkseg>

<trkpt lat="46.0988580" lon="14.5596710">

<ele>293.6</ele>

<time>2023-12-16T14:24:24Z</time>

<extensions>

<power>404</power>

<gpxtpx:TrackPointExtension>

<gpxtpx:atemp>25</gpxtpx:atemp>

<gpxtpx:hr>140</gpxtpx:hr>

<gpxtpx:cad>82</gpxtpx:cad>

</gpxtpx:TrackPointExtension>

</extensions>

</trkpt>

<trkpt lat="46.0988890" lon="14.5596830">

<ele>293.6</ele>

<time>2023-12-16T14:24:25Z</time>

<extensions>

<power>408</power>

<gpxtpx:TrackPointExtension>

<gpxtpx:atemp>25</gpxtpx:atemp>

<gpxtpx:hr>141</gpxtpx:hr>

<gpxtpx:cad>81</gpxtpx:cad>

</gpxtpx:TrackPointExtension>

</extensions>

</trkpt>

<trkpt lat="46.0989220" lon="14.5597030">

<ele>293.8</ele>

<time>2023-12-16T14:24:26Z</time>

<extensions>

```

The svg files in which we store images in vector format are XML.

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

ARSO publishes the weather forecast in XML format. This is probably friendly to the authors of the various phone apps that capture data from their website.

Even if we rename an .xlsx or .docx file to zip and unzip it, we find XMLs containing our text and its formatting.

Sitcoms

We got a list with the most popular sitcoms in Python format from ChatGPT.

<https://www.omdbapi.com/> - here you get the posters from

The queries are of the form http://www.omdbapi.com/?apikey={api\_key}&{parameters}. You can search by titles and whatnot, most useful is to search by imdb codes. They are easy to get



Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

split the imdb\_id with split("/") and take the second to last part - the last part is empty because there is a slash at the end.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

We imported the urlopen function from the urllib.request module. We give it an URL as an argument and it returns the server’s response. The response contains a lot of things, like

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

Automatisch generierte Beschreibung

* Which can be 404 (if the page doesn’t exist), 500 (if there’s something wrong with it) and 418 (if we happen to be talking to a teapot). Here it is 200, which is OK.

There are more things we can find out:

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

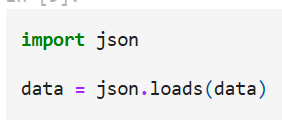
* But none of this interests us.

All we need to know here is that response has a read method that returns the data we have been given. We've already extracted them above, so (as with files) they are consumed. Right, let's retrieve them again and save them this time.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

The data looks like a Python dictionary. It's actually JSON, which is, um, roughly a JavaScript dictionary. The most reliable way to convert them to a Python dictionary is json.loads.



Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

I said I needed pictures. I got a lot of useful stuff, but for now: a picture. This one is obviously in

Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

Well, not the image, its URL.

If I type <img src="https://m.media-amazon.com/images/M/MV5BOTU2YmM5ZjctOGVlMC00YTczLTljM2MtYjhlNGI5YWMyZjFkXkEyXkFqcGc@.\_V1\_SX300.jpg"/> into this Jupyter, it will show it.

I would, of course, want to pick it up in a file.

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Whatever is in this poster, it is obviously a picture. Let's save it in a file whose name will be the name of the batch.

Ein Bild, das Text, Schrift, Reihe, weiß enthält.

Automatisch generierte Beschreibung

Don't overlook the second argument when opening a file: wb. w tells us, as we already know, that we want to write to the file. b tells us that it is a binary file.

I look in the directory and, lo and behold, there is indeed a "Friends.jpg" file with a poster of the series (which, disclaimer, I never watched; even though I was just the right age, it seemed a bit ... saccharine).

Satisfied with the result, let's put it all together in a finished program.



A minor annoyance: on Windows and Linux, certain characters are banned from filenames:

Before writing a file, we need to correct the name accordingly.

**Conclusion**

Two things we have not discussed. The first is broad, very broad: how to send more complex requests, work with cookies and so on. And how to better understand the answers we get from the server. This is in the area of web application development, we are not really going to deal with that here.

The second is what was that poster. You may have noticed that it is preceded by a b:

Ein Bild, das Text, Schrift, weiß, Reihe enthält.

Automatisch generierte Beschreibung

We will deal with this in the next topic.