Theory: Files in Python

14 minutes 7 / 7 problems solved

Start practicing

Often data for your program comes from the outside in the form of files. We all know what files are, we all use them in our everyday lives so it makes sense to incorporate them in a project. In this topic, you'll learn how to work with external files.

The general scheme of working with files is the following: open the file, do what you need with it, then close it. Let's consider these steps in more detail.

§1. Open file

Before we can try to do anything with a file, we need to open it. Python has a lot of built-in functions for working with files so we don't even need to install or import any modules. To open a file, we can use the built-in function `open()`.

```
# example of opening a file
my_file = open('my_file.txt')
```

The `open()` function has one required parameter `file` which is a **path-like object**. The path-like object is a `str` or `bytes` that represents a path in the file directory. In our example, the `file` parameter has the value `"my_file.txt"` which means that the file "**my_file.txt**" lies in the current working directory. In this topic, we won't get into details about paths and directories, so we'll just assume that all our example files lie in the current working directory.

The object `my_file` we've just created is a **file** or **file-like object**. It just means that we can use different kinds of file methods on this object.

The `open()` function has a number of optional parameters. If you take a look at the official Python documentation, you can learn more about them. In this topic, we'll look at two parameters: `mode` and `encoding`.

§2. The mode parameter

One of the most important optional parameters of the `open()` function is `mode`. This parameter regulates how we want to open our file and what for. The following options are available:

`'r'`	Open for reading. If the file doesn't exist, an error occurs.
`'w'`	Open for writing and truncate. If the file already exists, it will be overwritten.
`'a'`	Open for writing. If the file already exists, append to the end of the file.
`'b'`	Open in binary mode.
`'+'`	Open for updating (reading and writing).
`'t'`	Open as a text

Access modes

A couple of things need to be said about modes.

First of all, by default, files are opened for reading as a text, so the default value of `mode` is `'r'` or, more precisely, `'rt'`.

Second, as you can see, we can combine modes to do what we need. For example, if we want to open an existing file and be able to read and update it, we should set the mode as `'r+'`.

Third, we can choose the format in which we want to open files. The main options are text or binary, `'t'` and `'b'` respectively. This corresponds to the difference between `str` and `bytes` objects. So, if you want to open a file for

Current topic:
Files in Python Stage 2

Topic depends on:
Files Stage 2
Invoking a function Stage 1 16

Topic is required for:
Reading files Stage 2
Writing files Stage 2

writing in binary, the mode should be `'wb'`. Note that since text format is the default format, most of the times `'t'` is omitted.

> Some modes cannot be combined with each other: thus, only one of the options `'w'`, `'r'`, and `'a'` must be specified, we can't open a file with `'ar'` mode. Similarly, we must choose either `'b'` or `'t'`, the file can't be opened both in the text and binary modes.

Lastly, we should mention an important difference between options `'w'` and `'a'`. Both these modes are used for writing to a file. The only difference is that `'w'` truncates the file before writing to it. In other words, if the file already exists, its contents are deleted. The `'a'` behaves differently: if the file exists, anything that we write to it will be simply added to the end of the file.

§3. Encoding

The `encoding` parameter specifies the encoding that should be used to decode or encode the text file. It is needed when we open the file as a text, and the default value depends on the platform. Below are some examples of opening files for reading in different encodings:

```
# UTF-8
file_utf8 = open('my_file.txt', encoding='utf-8')

# UTF-16
file_utf16 = open('my_file.txt', encoding='utf-16')

# CP1252
file_cp1252 = open('my_file.txt', encoding='cp1252')
```

You may know that if you're not using the right encoding on your file, the information will not make sense at all. So, if the file you're trying to open looks weird or wrong, sometimes it makes sense to fiddle with this parameter.

§4. Closing the file

After working with a file, we need to close it. Closing files is extremely important! In most cases, a file will be closed eventually when the program finishes working. However, there is no guarantee for that. In order to ensure the safety of the data, we must make sure that the file is closed in the end. One of the ways to do it is by using the `close()` method on the file:

```
# closing the file
my_file.close()
```

This is not the only way, nor the most efficient. For now, it'll do and you'll learn about other ways in another topic.

§5. Summary

In this topic, you've learned the basics of working with files in Python: how to open and close them. When opening files, you need to specify what you are opening them for, for example, reading or writing. Sometimes we also need to specify the encoding of the file so that our data can be properly encoded/decoded.

In the following topics, you'll learn about what you can actually do with the file and how it can be done!

Report a typo

358 users liked this theory. **7** didn't like it. **What about you?**

Start practicing

Comments (6) Hints (0) Useful links (3) Show discussion