Theory: Context manager

20 minutes 8 / 8 problems solved

We live in a world of limited resources, so one of the most important skills in life (and in programming) is knowing how to manage them. We cannot teach you how to manage your resources in real life, but we can help you effectively manage resources in Python with the help of **context managers.**

§1. When to use context managers

The main purpose of context managers is, as you might've guessed, resource management. What does this mean in practice? The most common example is opening files. Opening a file consumes a limited resource called a **file descriptor.** If you try to open too many files at once, depending on your operating system, you may get an error or completely crash your program.

```
# don't try this at home!
n_files = 1000000
files = []

for i in range(n_files):
    files.append(open('test.txt'))

# OSError: [Errno 24] Too many open files
```

To avoid file descriptor **leakage** (as presented above), we need to close the files after we're done with them. Closing the files is done with the `close()` method.

```
n_files = 1000000
files = []

for i in range(n_files):
    f = open('test.txt')
    files.append(f)
    f.close()

# no errors, all correct!
```

This works perfectly fine if we have relatively simple programs. However, as our programs or our file manipulations get more complicated, determining when and how to close the files may get tricky. In other programming languages, a common way to deal with this is a `try ... except ... finally` block. In Python, we can use a context manager. Basically, the context manager guarantees that all necessary operations will take place at the right time. In the example with opening files, the context manager will close the file and release the file descriptor when we are done working with the file.

§2. with ... as

Now that we know why we need to use context managers, let's learn how to do that. A context manager is introduced by a `with` keyword followed by the context manager itself and the name of the variable. The basic syntax is the following:

```
# invoking a context manager
with statement as variable_name:
    ...
```

`statement` here is anything that acts like a context manager (meaning, it supports specific context manager methods). It can be either a custom made context manager or Python's internal one. The following objects are some of the context managers in Python:

- File objects (and other streams like `io.StringIO` or `io.BytesIO`);
- Sockets;

Current topic:
Context manager Stage 2

Topic depends on:
Reading files Stage 2
Writing files Stage 2

Topic is required for:
Working with CSV
Socket module
Json module Stage 2

- Locks and semaphores in the `threading` module;
- Database connections;
- Mock objects.

We can also nest this construction:

```
# nested context manager
with statement1 as var1:
    with statement2 as var2:
        # and so on
        ...
```

Most commonly, `with ... as` statement is used when working with files. Let's see how we can do that.

§3. Working with files

A file object that we get when we use the `open()` function acts as a context manager, so we can use it as the `statement` part of the code. This is how it can be done:

```
with open('test.txt') as f:
    # work with the file
    ...
```

As you can see, it is very simple! It also allows us to shorten our code a little since we don't need to explicitly close the file at the end.

> Note that you CAN explicitly close the file within the `with ... as` statement, it won't be an error. You just don't need to!

Coming back to the situation with a million files, this is how it looks if we use context manager:

```
n_files = 1000000
files = []

for i in range(n_files):
    with open('test.txt') as f:
        files.append(f)

# OK!
```

Now, let's look at a more realistic example. Suppose, we have a file with the movies directed by Quentin Tarantino named *tarantino.txt*. We want to read this file and print the titles:

```
with open('tarantino.txt', 'r', encoding='utf-8') as f:
    for line in f:
        # we use strip() to get rid of newline symbols
        print(line.strip())
```

We'll get the following output:

```
Reservoir Dogs
Pulp Fiction
Jackie Brown
Kill Bill: Volume 1
Kill Bill: Volume 2
Grindhouse: Death Proof
Inglorious Basterds
Django Unchained
The Hateful Eight

Once Upon a Time in Hollywood
```

Now, imagine that we want to process these titles, say, make them all lowercase, and have it saved to a file. Here's how it can be done:

```
with open('tarantino.txt', 'r', encoding='utf-8') as in_file, \
     open('tarantino_lowercase.txt', 'w', encoding='utf-8') as out_file:
    for line in in_file:
        out_file.write(line.lower())
```

The file *tarantino_lowercase.txt* that we've created in the process, will contain the titles of Tarantino movies written in lowercase.

§4. Summary

In this topic, we've learned about context managers, special structures used for effective managing of resources. Basically, context managers help make sure that all necessary operations have been carried out. Context managers are usually introduced by a `with ... as` statement.

In practice, you'll most commonly encounter this in opening files. However, you can also create custom context managers for your own purposes, but this is a skill for another topic!

Report a typo

272 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (8) Hints (0) Useful links (0) Show discussion

Table of contents: