

LECTURE 9

CONTEXT MANAGERS

MCS 275 Spring 2021
Emily Dumas

LECTURE 9: CONTEXT MANAGERS

Course bulletins:

- Quiz 3 is due Tuesday at Noon.
- Project 1 due Friday at 6pm CST.
- Project 1 autograder is available.

MOTIVATING EXAMPLE

Here's a common way to deal with file input/output:

```
fileobj = open("data.txt", "w")  
fileobj.write(...)  
# other write operations...  
fileobj.close()
```

MOTIVATING EXAMPLE

Here's a common way to deal with file input/output:

```
fileobj = open("data.txt", "w") # SETUP: Acquire file access
fileobj.write(...)
# other write operations...
fileobj.close()
```

MOTIVATING EXAMPLE

Here's a common way to deal with file input/output:

```
fileobj = open("data.txt", "w") # SETUP: Acquire file access
fileobj.write(...)
# other write operations...
fileobj.close() # CLEANUP: Release file access
```

POSSIBLE BUG

It is easy to forget to close a file, especially when the work after opening it involves conditionals, loops, return from a function, possible exceptions, etc.

Moreover, it can be hard to check whether a file is always closed when no longer in use.

All files are closed when a program exits, but open files are a limited resource. In long-running programs, holding on to many open files can be a problem.

Will this function always close the file?

```
def file_contains_walrus(fn):  
    """Return True if "walrus" is a line of file `fn`"""  
    fileobj = open(fn, "r")  
    for line in fileobj:  
        if line.strip() == "walrus":  
            fileobj.close()  
            return True  
    return False
```

Currently, in CPython (the usual interpreter): **Yes.**

In CPython, local variables are deleted as soon as a function returns. Deleting a file object closes the file.

But this isn't a language guarantee!

ANOTHER WAY

Use **with** block to ensure automatic file closing, and to be explicit about what part of a program needs the file.

```
with open("data.txt", "w") as fileobj:  
    fileobj.write(...)  
    # other write operations...  
print("At this point, the file is already closed")
```

Notice that you can see exactly what part of the program uses the file.

CLEANUP GUARANTEE

A file opened using a `with` block will be closed as soon as execution leaves the block, even if an exception is raised.

RECOMMENDATION

Always open files using `with`, and make the body as short as possible.

Think of files like refrigerators: Open them for the shortest time possible, and don't forget to close them!

CONTEXT MANAGERS

`with` is not a Python language feature created solely for files.

Any object that is a **context manager** can be used.

A context manager is any object that defines special methods to:

- Perform setup (`__enter__`)
- Perform cleanup (`__exit__`)

PURPOSE OF CONTEXT MANAGERS

Context managers are appropriate when the creation or use of an object will take control of a resource that later needs to be released, e.g.

- Network connections
- Database connections
- Locks
- Any limited or exclusive access right

CONTEXT MANAGER PROTOCOL

An object is a context manager if it has methods:

- `__enter__(self)` : Performs setup; return value assigned to the name after "as" (if any)
- `__exit__(self, exc_type, exc, tb)` : Perform cleanup. The arguments describe any exception that happened in the `with` block that is the reason for the exit (or `None` if no exception happened).

BUILT-IN CONTEXT MANAGERS

We've seen that file objects (created by `open()`) are context managers.

A `threading.Lock` is also a context manager; setup will acquire the lock, and cleanup will release it, e.g.

```
L = threading.Lock()
# Do things not requiring exclusive access
with L:
    print(shared_dict["name"])
    print(shared_dict["address"])
# Back to non-exclusive stuff.
```

Note that we use `with` without `as` in this case.

REFERENCES

- *Lutz* discusses context managers in Chapter 34. This is a long chapter covering several other topics. Look for the heading **with/as Context Managers**. In the print edition, it beings on page 1114.

REVISION HISTORY

- 2021-02-01 Initial publication

