

# LECTURE 8

## CONTEXT MANAGERS

MCS 275 Spring 2023  
Emily Dumas

# LECTURE 8: CONTEXT MANAGERS

Reminders and announcements:

- Project 1 due Fri Feb 10 at 6pm central.
- Project 1 autograder will open soon.
- Homework 4 will be posted Thursday night.

# PUZZLE

What's the output?

```
A = 2
B = [3, 4, 5]
C = [5, 6, 7]

def f():
    A = 7
    B[0] = 7
    C = [7, 7, 7]

f()
print(A)
print(B[0])
print(C[0])
```

# PUZZLE

What's the output?

```
A = 2
B = [3,4,5]
C = [5,6,7]

def f():
    A = 7          # new local A
    B[0] = 7       # B.__setitem__(0,7) on global B
    C = [7,7,7]    # new local C

f()
print(A)          # 2    (unchanged)
print(B[0])       # 7    (same B, new item at index 0)
print(C[0])       # 5    (unchanged)
```

# MARKDOWN

Text cells (Colab) or markdown cells (Jupyter) contain formatted text. When editing, formatting is specified with a language called Markdown.

```
# Heading level 1
## Heading level 2
### Heading level 3
```

```
* Bullet list item
* Another bullet list item
```

```
1. Numbered list item
1. Another numbered list item
```

```
Links: [text to display](https://example.com)
```

# COMMON PATTERN

Acquire, use, and free a resource.

```
x = resource() # open file, connect to database, ...  
  
x.action()  
x.action2(y,z)  
if x.status() == w:  
    ...  
  
x.close() # or "release" or "delete"
```

# EXAMPLE: FILE I/O

```
# ACQUIRE
fp = open("data.txt", "w", encoding="UTF-8")
# USE
for s in L:
    fp.write(s+"\n")
# RELEASE
fp.close()
```

# POSSIBLE BUG

Is the resource always freed? What if an exception is raised?

All files are closed when a program exits, but open files are a limited resource.



# 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", encoding="UTF-8")  
    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.

```
with open("data.txt", "w", encoding="UTF-8") as fileobj:  
    fileobj.write(...)  
    fileobj.write(...)  
    # other write operations...  
print("At this point, the file is already closed")
```

Extra bonus: you can see exactly what part of the program needs the open 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.

# IN OTHER LANGUAGES

Other OO languages often recommend RAI:  
**R**esource **A**cquisition **i**s **I**ntantiation.

Making an instance of a class acquires a resource,  
which is held for the lifetime of the object.

The resource is then freed by the class destructor when  
the object is deleted.

# PYTHON OBJECT LIFETIME

Python deletes objects you can no longer access (*garbage collection*) but:

- No promises about exactly when
- No guarantee any function (destructor) gets called as part of deletion
- Manual deletion is discouraged

Thus Python's `with` blocks are a substitute for RAII.

# CONTEXT MANAGERS

Any object whose class is a **context manager** can be used in a `with`-block.

A context manager is a class with special methods:

- `__enter__` to perform setup
- `__exit__` to perform cleanup



# EXAMPLES

Context managers are appropriate for:

- Network connections
- Database connections
- Locks
- Any limited or exclusive access right
- Temporary setup or changes that must be reverted

# CONTEXT MANAGER PROTOCOL

- `__enter__(self) :`
  - Performs setup
  - Return value is assigned to the name after `as` in `with` statement.
- `__exit__(self, exc_type, exc, tb) :`
  - Performs cleanup.
  - The arguments describe any exception that happened in the `with` block.

Expect each method to be called exactly once.

# BUILT-IN CONTEXT MANAGERS

Some examples (listed as *class - resource*)

- `open` - Open file
- `threading.Lock` - Thread-exclusive right
- `urllib.request.urlopen` - HTTP connection
- `tempfile.TemporaryFile` - Temporary file  
(deleted after use)

# REFERENCES

- [Python documentation on Context Manager types](#)
- *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

- 2022-01-31 Last year's lecture on this topic finalized
- 2023-01-30 Updated version for spring 2023

