CONTEXT MANAGERS

What is a context?

Oxford dictionary: The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood.

In Python: the state surrounding a section of code

```
# module.py
f = open('test.txt', 'r')
print(f.readlines())
f.close()
```

when `print(f.readlines())` runs, it has a context in which it runs

→ global scope

Managing the context of a block of code

Consider the open file example:

```
# module.py

f = open('test.txt', 'r')
perform_work(f)
f.close()
```

There could be an exception before we close the file → file remains open!

Need to better "manage" the context that `perform_work(f)` needs

```
f = open('test.txt', 'r')

try:
    perform_work(f)
finally:
    f.close()
```

this works → writing `try/finally` every time can get cumbersome

→ too easy to forget to close the file

Context Managers

→ create a context (a minimal amount of state needed for a block of code)

→ execute some code that uses variables from the context

→ automatically clean up the context when we are done with it

→ enter context

→ work within context

→ exit context

→ open file

→ read the file

→ close the file

Example

```
with open('test.txt', 'r') as f:
    print(f.readlines())
```

create the context → open file

work inside the context

exit the context → close file

Context managers manage data in our scope
→ on entry
→ on exit

Very useful for anything that needs to provide Enter / Exit Start / Stop Set / Reset

→ open / close file
→ start db transaction / commit or abort transaction
→ set decimal precision to 3 / reset back to original precision