

## Context Managers

You've probably already worked with opening and closing files.

```
In [ ]: f = open('dummy-file.txt', 'r')
        for line in f:
            print(line)

        f.close()      # People will always forget this!
```

```
In [ ]: with open('dummy-file.txt', 'r') as f:
        for line in f:
            print(line)

        # No need to close file since that's taken care of automatically
```

The keyword `with` marks a context. We can define our own [context managers](#) as well!

The idea is to create a context that requires some [setup](#) before starting and then some [cleanup](#) at the end.

```
In [51]: import time

        def some_function():
            time.sleep(1)
```

```
In [52]: %time some_function()      # but this is only available in jupyter notebook
```

```
CPU times: user 727 µs, sys: 1.76 ms, total: 2.49 ms
Wall time: 1.01 s
```

```
In [ ]: start_time = int(round(time.time() * 1000))

        some_function()

        end_time = int(round(time.time() * 1000))
        elapsed = end_time - start_time
        print("Code took %d ms to run." % elapsed)
```

Fairly simple but it's making our code look ugly and is a hassle. Let's do this using a context manager. It matches the pattern after all:

- Do something at startup (Record start time)
- Perform some (unspecified) work
- Do something at end (Report time elapsed)

```
In [53]: from contextlib import contextmanager # import the decorator

        @contextmanager
        def timeit():
            start_time = int(round(time.time() * 1000))

            yield      # Remember this guy from the generators lecture?

            end_time = int(round(time.time() * 1000))
            elapsed = end_time - start_time
            print("Code took %d ms to run." % elapsed)
```

Beyond Basic Programming - Intermediate Python

recluze.net/learn

```
In [54]: with timeit():  
         some_function()
```

Code took 1003 ms to run.

```
In [55]: def another_function():  
         time.sleep(0.5)
```

```
In [56]: with timeit():  
         another_function()
```

Code took 505 ms to run.

Clean code, saves time!

## Case Study: Temporary Directories

It's common to create temporary directories for files and delete them after you're done with them.

```
In [57]: import tempfile  
         import shutil  
         import os  
  
         try:  
             name = tempfile.mkdtemp()  
             print("Created temp directory: %s" % name)  
  
             filename = os.path.join(name, "somefile.txt")  
  
             # Do some processing  
             with open(filename, 'w') as f:  
                 print("Opened file: %s" % filename)  
                 f.write("Dummy text")  
  
         finally:  
             print("Deleting directory: %s" % name)  
             shutil.rmtree(name)
```

Created temp directory: /var/folders/sw/ch9b3fc964937k5cr4ff57l00000gn/T/tmp8pxal9tm  
Opened file: /var/folders/sw/ch9b3fc964937k5cr4ff57l00000gn/T/tmp8pxal9tm/somefile.txt  
Deleting directory: /var/folders/sw/ch9b3fc964937k5cr4ff57l00000gn/T/tmp8pxal9tm

```
In [ ]: @contextmanager  
         def tmpdir(filename):  
             try:  
                 name = tempfile.mkdtemp()  
                 print("Created temp directory: %s" % name)  
  
                 filename = os.path.join(name, filename)  
  
                 # Do some processing  
                 with open(filename, 'w') as f:  
                     print("Opened file: %s" % filename)  
                     yield f  
  
             finally:  
                 print("Deleting directory: %s" % name)  
                 shutil.rmtree(name)
```

```
In [ ]: with tmpdir('xyz') as f:  
         print("Doing something with the file ... ")
```

