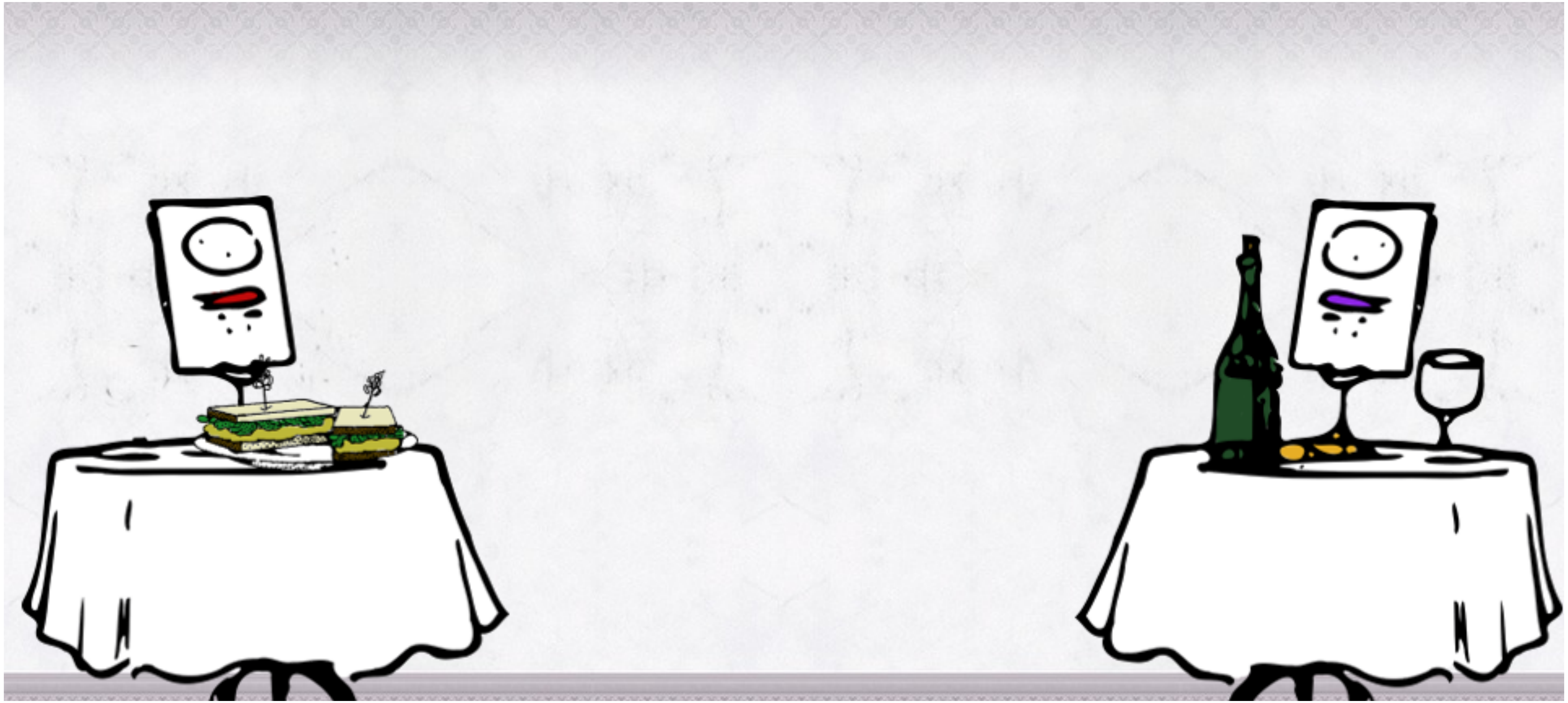
What is a context manager?

A context manager:

- Sets up a context
- Runs your code
- Removes the context

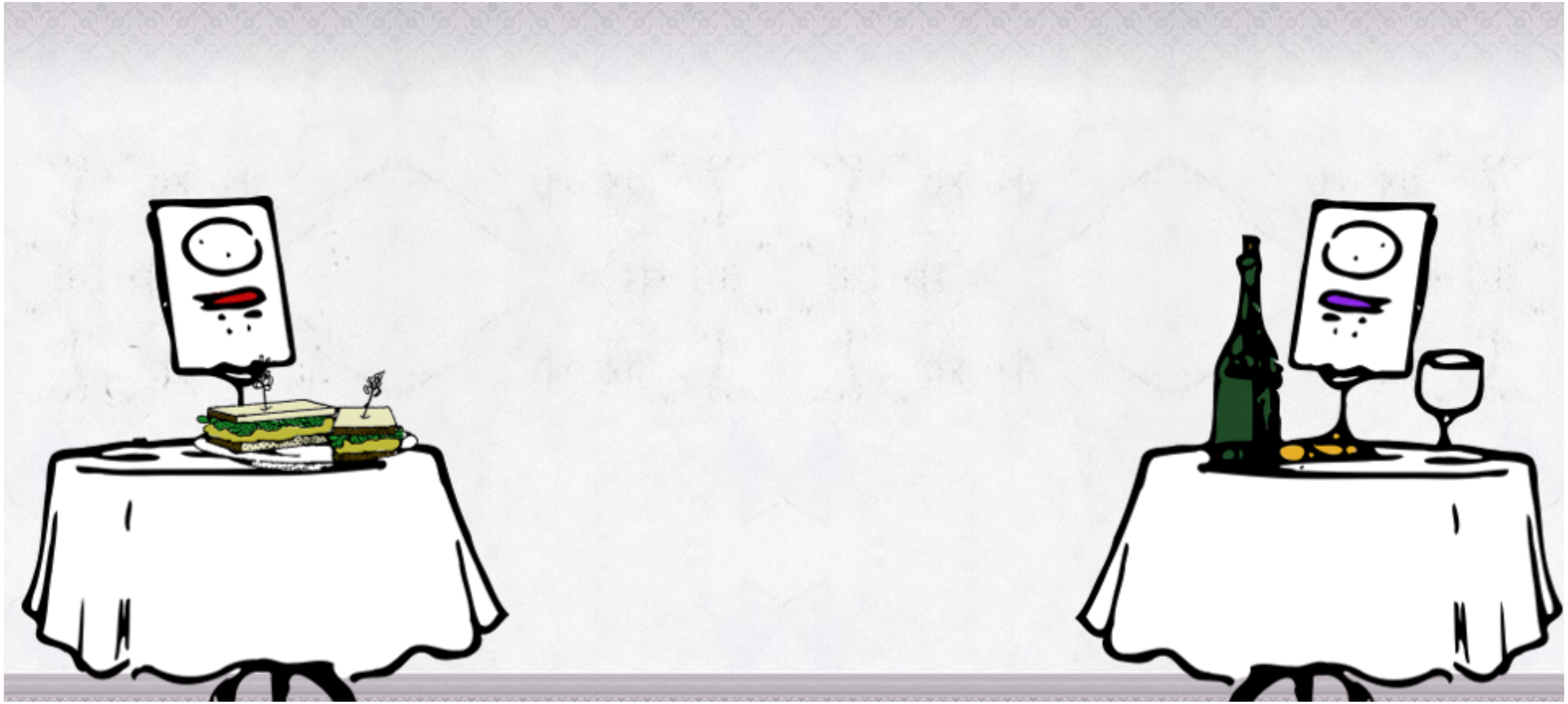
A catered party

A catered party



A catered party

A catered party



A catered party

Catered party as context

Context managers:

- Set up a context
- Run your code
- Remove the context

Caterers:

- Set up the tables with food and drink
- Let you and your friends have a party
- Cleaned up and removed the tables

A real-world example

```python
with open('my_file.txt') as my_file:
  text = my_file.read()
  length = len(text)

print('The file is {} characters long'.format(length))
```

`open()` does three things:

- Sets up a context by opening a file
- Lets you run any code you want on that file
- Removes the context by closing the file

Using a context manager

```
with
```

Using a context manager

```
with <context-manager>()
```

Using a context manager

```
with <context-manager>(<args>)
```

Using a context manager

```
with <context-manager>(<args>):
```

Using a context manager

```
with <context-manager>(<args>):
  # Run your code here
  # This code is running "inside the context"
```

Using a context manager

```
with <context-manager>(<args>):
  # Run your code here
  # This code is running "inside the context"

# This code runs after the context is removed
```

Using a context manager

```
with <context-manager>(<args>) as <variable-name>:
  # Run your code here
  # This code is running "inside the context"

# This code runs after the context is removed
```

```
with open('my_file.txt') as my_file:
  text = my_file.read()
  length = len(text)

print('The file is {} characters long'.format(length))
```

Let's practice!

WRITING FUNCTIONS IN PYTHON

Two ways to define a context manager

- Class-based
- Function-based

Two ways to define a context manager

- Class-based
- **Function-based** *

How to create a context manager

```python
def my_context():
  # Add any set up code you need
  yield
  # Add any teardown code you need
```

1. Define a function.
2. (optional) Add any set up code your context needs.
3. Use the "yield" keyword.
4. (optional) Add any teardown code your context needs.

How to create a context manager

```
@contextlib.contextmanager
def my_context():
  # Add any set up code you need
  yield
  # Add any teardown code you need
```

1. Define a function.
2. (optional) Add any set up code your context needs.
3. Use the "yield" keyword.
4. (optional) Add any teardown code your context needs.
5. Add the `@contextlib.contextmanager` decorator.

The "yield" keyword

```python
@contextlib.contextmanager
def my_context():
  print('hello')
  yield 42
  print('goodbye')
```

```python
with my_context() as foo:
  print('foo is {}'.format(foo))
```

```
hello
foo is 42
goodbye
```

Setup and teardown

```python
@contextlib.contextmanager
def database(url):
  # set up database connection
  db = postgres.connect(url)

  yield db

  # tear down database connection
  db.disconnect()
```

```python
url = 'http://datacamp.com/data'
with database(url) as my_db:
  course_list = my_db.execute(
    'SELECT * FROM courses'
  )
```

Setup and teardown

```python
@contextlib.contextmanager
def database(url):
  # set up database connection
  db = postgres.connect(url)

  yield db

  # tear down database connection
  db.disconnect()
```

```python
url = 'http://datacamp.com/data'
with database(url) as my_db:
  course_list = my_db.execute(
    'SELECT * FROM courses'
  )
```

Setup and teardown

```
@contextlib.contextmanager
def database(url):
  # set up database connection
  db = postgres.connect(url)

  yield db

  # tear down database connection
  db.disconnect()
```

```
url = 'http://datacamp.com/data'
with database(url) as my_db:
  course_list = my_db.execute(
    'SELECT * FROM courses'
  )
```

Yielding a value or None

```python
@contextlib.contextmanager
def database(url):
  # set up database connection
  db = postgres.connect(url)

  yield db

  # tear down database connection
  db.disconnect()
```

```python
url = 'http://datacamp.com/data'
with database(url) as my_db:
  course_list = my_db.execute(
    'SELECT * FROM courses'
  )
```

```python
@contextlib.contextmanager
def in_dir(path):
  # save current working directory
  old_dir = os.getcwd()

  # switch to new working directory
  os.chdir(path)

  yield

  # change back to previous
  # working directory
  os.chdir(old_dir)
```

```python
with in_dir('/data/project_1/'):
  project_files = os.listdir()
```

Let's practice!

WRITING FUNCTIONS IN PYTHON

Nested contexts

```
def copy(src, dst):
  """Copy the contents of one file to another.

  Args:
    src (str): File name of the file to be copied.
    dst (str): Where to write the new file.
  """
  # Open the source file and read in the contents
  with open(src) as f_src:
    contents = f_src.read()

  # Open the destination file and write out the contents
  with open(dst, 'w') as f_dst:
    f_dst.write(contents)
```

Nested contexts

```python
with open('my_file.txt') as my_file:
  for line in my_file:
    # do something
```

Nested contexts

```
def copy(src, dst):
  """Copy the contents of one file to another.

  Args:
    src (str): File name of the file to be copied.
    dst (str): Where to write the new file.
  """
  # Open both files
  with open(src) as f_src:
    with open(dst, 'w') as f_dst:
      # Read and write each line, one at a time
      for line in f_src:
        f_dst.write(line)
```

Handling errors

```python
def get_printer(ip):
  p = connect_to_printer(ip)

  yield

  # This MUST be called or no one else will
  # be able to connect to the printer
  p.disconnect()
  print('disconnected from printer')

doc = {'text': 'This is my text.'}

with get_printer('10.0.34.111') as printer:
  printer.print_page(doc['txt'])
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    printer.print_page(doc['txt'])
KeyError: 'txt'
```

Handling errors

```
try:
  # code that might raise an error
except:
  # do something about the error
finally:
  # this code runs no matter what
```

Handling errors

```python
def get_printer(ip):
  p = connect_to_printer(ip)

  try:
    yield
  finally:
    p.disconnect()
    print('disconnected from printer')

doc = {'text': 'This is my text.'}

with get_printer('10.0.34.111') as printer:
  printer.print_page(doc['txt'])
```

```
disconnected from printer
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    printer.print_page(doc['txt'])
KeyError: 'txt'
```

Context manager patterns

Open	Close
Lock	Release
Change	Reset
Enter	Exit
Start	Stop
Setup	Teardown
Connect	Disconnect

[1] Adapted from Dave Brondsema's talk at PyCon 2012: https://youtu.be/cSbD5SKwak0?t=795

Let's practice!

WRITING FUNCTIONS IN PYTHON