Page Objects allow greater abstraction in test cases

```
class AdvancedYoutubeTests(object):

    @pytest.mark.parametrize('video', [vids[v] for v in vids], ids=[v for v in vids])
    def test_search_by_vid(self, driver, video):
        “”” For a given video, see if you can find it by search. “””

        # setup
        id = video['vid']  # extract the id from the parametrized data input
        this_video = VideoDetailObject(id)  # load the page object model

        # load the home page
        page = HomePageObject(driver)

        # perform the search
        assert page.search(driver, id)
```

Component: Application Wrappers continued

You can treat an API in pretty much the same way, using an Endpoint Object Model build on top of an HTTP library like requests.

Let's build an EOM for our pretend custom lettering API.

```
my_test_framework
    |- my_test_framework
        |- apps
            |- website
                - base.py
                - home.py
                - cart.py
            |- api
                - base.py
                - letters.py
        |- data
            - shirts.py
        |- framework
            - selenium_utils.py
            - utils.py
            - exceptions.py
        |- tests
            - conftest.py
            - pytest.ini
            - test_simple.py
    - requirements.txt
```

Page Objects allow greater abstraction in test cases

```
class AdvancedYoutubeTests(object):

    @pytest.mark.parametrize('video', [vids[v] for v in vids], ids=[v for v in vids])
    def test_search_by_vid(self, driver, video):
        """ For a given video, see if you can find it by search. """

# setup
id = video['vid']  # extract the id from the parametrized data input
this_video = VideoDetailObject(id)  # load the page object model

# load the home page
page = HomePageObject(driver)

# perform the search
assert page.search(driver, id)
```