Custom Exceptions Support Functional Tests

```python
class UnexpectedStatusCodeException(Exception):
    """
        Raise this exception when the server response code from an http request is not
        a success code.
        Capture the requests response object and make that available.

        # pass the response object to the exception
        >>> raise UnexpectedStatusCodeException(response=res)

        # then catch and introspect the exception's property
        >>> except UnexpectedStatusCodeException as e:
            ... print(e.response.status_code)
        400
    """

    def __init__(self, response):
        Exception.__init__(self, response)
        self.response = response


class PageIdentityException(Exception):
    """
        Raise this exception when a page fails its self validation of identity.
    """

    pass
```

Plumbing

Everything we've discussed so far has been the basic structure setup for the meat of test automation: business process abstractions that support end-to-end automation.

Our app wrappers support extensive deep functional tests. Our business abstractions will let us define happy paths through our functional affordances so that we can focus on user journeys.

Custom Exceptions Support Functional Tests

```python
class UnexpectedStatusCodeException(Exception):
    """

    Raise this exception when the server response code from an http request is not
    a success code.
    Capture the requests response object and make that available.

    # pass the response object to the exception
    >>> raise UnexpectedStatusCodeException(response=res)

    # then catch and introspect the exception's property
    >>> except UnexpectedStatusCodeException as e:
        ... print(e.response.status_code)
    400
    """
    def __init__(self, response):
        Exception.__init__(self, response)
        self.response = response


class PageIdentityException(Exception):
    """
    Raise this exception when a page fails its self validation of identity.
    """
    pass
```