Paquete/herramienta alternativa a Sphinx para documentación

Existen diversas herramientas desarrolladas para facilitar la tarea de documentación de los distintos proyectos de software. Sin embargo, sólo un pequeño grupo poseen el soporte una comunidad de desarrolladores y que se encuentren en activo desarrollo [1], [2].

Las herramientas identificadas han sido:

- `pdoc` Es una herramienta simple, diseñada para la documentación de proyectos escritos con `python`
- `pdoc3` Herramienta muy similar a `pdoc`, suele asociarse y tratarse como la misma herramienta pero no lo son [3]
- `Doxygen` Es una herramienta que trabaja con varios lenguajes de programación y muy usada en proyectos escritos en `C` + +
- `pydoctor` herramienta diseñada para trabajar con `python` 2

Adicionalmente, durante la investigación, se encontró que `python` posee una herramienta de documentación incluida en su core -`pydoc`-

Dicha herramienta es la responsable de generar la documentación de los módulos de `python` en la terminal a través de la función `help()` haciendo uso de los docstrings de los módulos, clases, funciones y métodos [4]

Es importante recalcar que `pydoc` no sólo generá documentación en la consola, sino que puede generar archivos `html` con la documentación o visualizarlo en un navegador de internet.

Por transitividad, todos nuestros módulos, clases, funciones y métodos pueden ser documentados con `pydoc` obteniendo el beneficio de poder mostrar nuestra documentación en la consola. Sin embargo, la visualización en `html` es muy cargada y no de las más amigables para los usuarios.

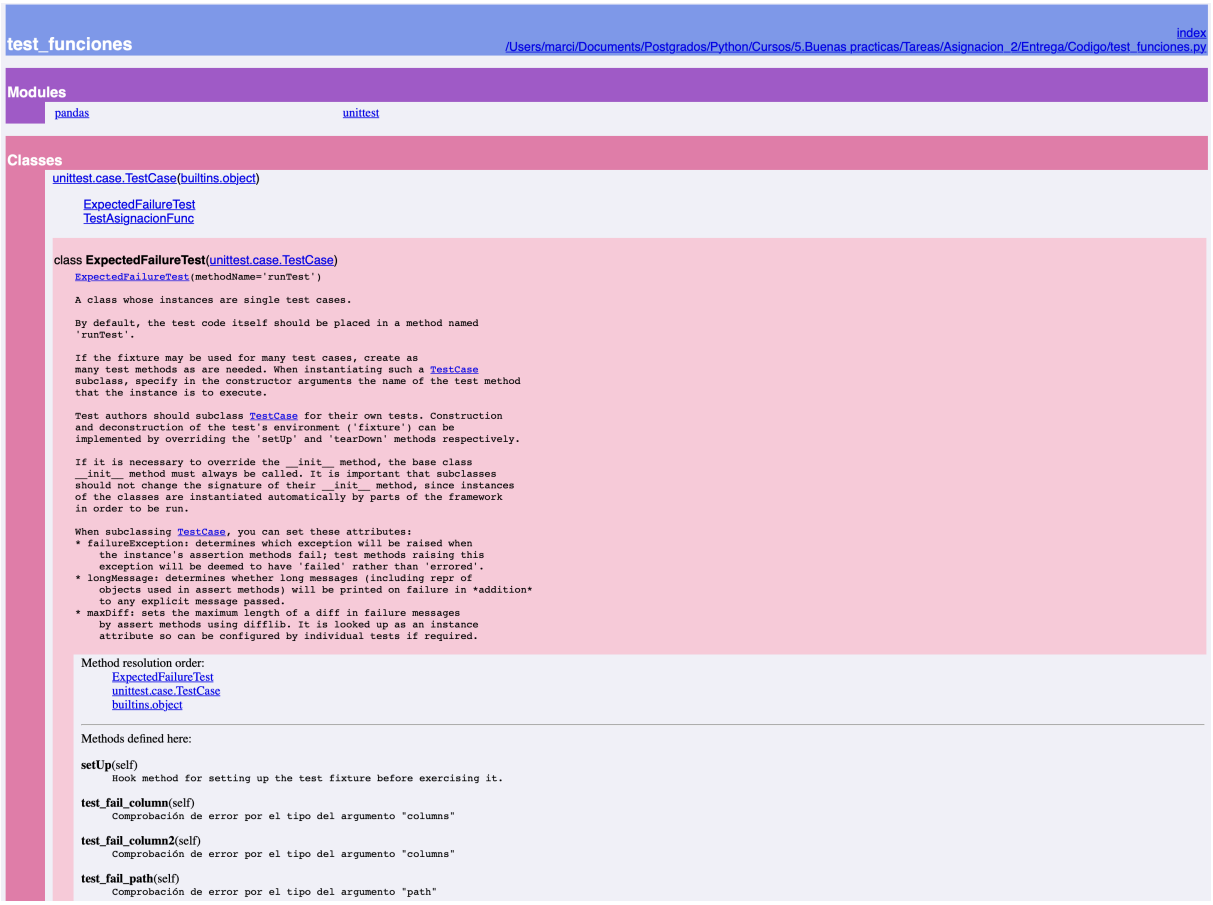

Figure 1: Documentación generada por pydoc

Trás un analísis de las opciones encontradas, se decidió seleccionar la herramienta `pdoc` para el desarrollo del manual debido a su simplicidad y posibilidad de migración a `sphinx` con mínimos esfuerzos.

Pdoc

Es una librería desarrollada para la generación automática de documentación de proyectos desarrollados con `python`, cuyo principio general es la simplicidad [5]

> "`pdoc` aims to do one thing and to do it well."

`pdoc` es compatible con `python 3.7` y nuevas versiones

Características

- Al igual que el resto de herramientas, `pdoc`, hace uso de los docstrings para generar la documentación
- Soporta los siguientes formatos de texto en docstrings:
  - `Markdown`
  - `rStructuredText`
  - `numpydoc`
  - `Google-style`
- Soporte de anotaciones y otras características modernas de `python`
- Los archivos `html` pueden ser customizables
- Salida `html` independiente sin dependencias adicionales
- Recargas automaticas del servidor web cada vez que se realiza una modiicación de los sripts
- Documentación de variables
- Es capaz de renderizar fórmulas Matemáticas de los docstrings
- Selección de variables, módulos, clases, funciones y/o métodos a documentar con el uso de la variable `__all__`

Desventajas

- No ofrece la versatilidad de `sphinx` para generar distintas extensiones y/o temas
- Puede ser usado para generar una librería de módulos, pero la página principal `index` debe ser editada desde su fichero `html`
- No es compatible con toda la sintaxis de `rStructuredText`, pero su comunidad trabaja en incluir nuevas funcionalidades continuamente
- En caso de querer excluir algunos parámetros (por ejemplo: `inherited-members`) es necesario importar la librería e incluir parámetros en nuestros scripts

> **Nota:** Al igual que el resto de herramientas, `pdoc` debe importar todos los módulos para buscar los objetos a documentar. Por lo tanto, cualquier segmento de código será ejecutado. Se recomienda hacer uso del siguiente comando para asegurar que el código sea ejecutado sólo cuando sea invocado el script y no cuando sea importado
>
> ```
> 1 if __name__ == '__main__':
> 2   ...
> ```

En caso de requerir una documentación más profesional o generar documentos más complejos la misma documentación de `pdoc` recomienda usar `sphinx` [3]

Ejemplo de implementación

Para nuestro caso partícular se procederá a generar la documentación de los módulos desarrollados para las pruebas unitarias creados para la Asignación #2 de BPP.

Adicionalmente, se usará el formato `rStructuredTest` para documentar nuestros docstrings debido a que se desea migrar el desarrollo a `sphinx` para la entrega de esta asignación.

Una vez nos aseguramos que los docstrings se encuentran debidamente documentados procedemos a ejecutar los siguientes pasos

Instalación

Procedemos a instalar el paquete `pdoc` a través del administrador de paquetes `pip`

```
1 pip install pdoc
```

Generación de la documentación

En version web

Para generar la documentación en localhost sólo debemos ejecutar el siguiente comando.

- En caso que sólo se desee generar la documentación de un sólo módulo

  ```
  1 pdoc nombre_del_modulo.py
  ```

- En caso de que se desee generar la documentación de distintos módulos como una librería

  ```
  1 pdoc nombre_del_modulo_1.py nombre_del_modulo_2.py
  ```

> **Nota:** Los comandos mostrados anteriormente suponen que hemos abierto la terminal en el directorio que posee los módulos a documentar
>
> En caso de no estar en el directorio correcto podemos dirigirnos a dicho directorio haciendo uso del comando `cd` en la terminal o es posible sustituir el `nombre_del_modulo.py` por el `path` correcto del mismo

Obtendremos una respuesta como la indicada en la imagen siguiente en nuestra terminal.

```
(env) marci@Marcials-MacBook-Pro:~/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_3/pdoc/code$
pdoc test_funciones.py funciones.py errors.py
pdoc server ready at http://localhost:8080
```

Figure 2: Respuesta de la terminal

En caso de querer generar la documentación en un servidor distinto al localhost podemos hacer uso del flag `-h` // `--host` e indicar el host como argumento.

`pdoc` seleccionará un puerto disponible para correr el servidor `HTTP`, en caso de querer indicar el puerto a usar podemos hacer uso del flag `-p` // `--port` e indicar el puerto como argumento.

Ejemplo

```
pdoc nombre_del_modulo.py -p 1234
```

Adicionalmente, observaremos que `pdoc` abrirá de forma autómatica la documentación en el navegador

- Vista de la documentación de un sólo módulo

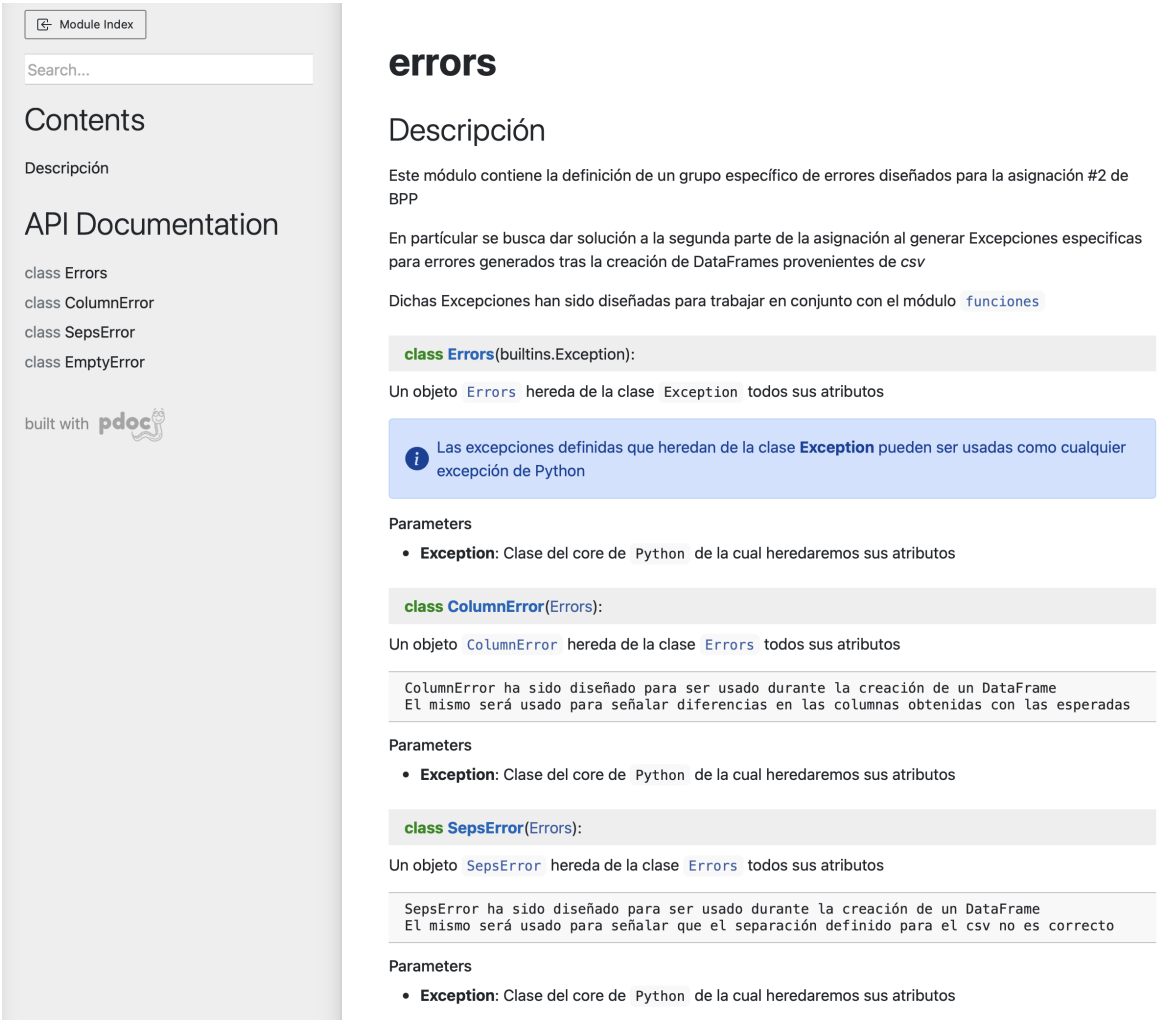

Figure 3: Documentación módulo

- Pantalla de inicio (índice) de la documentación para varios módulos

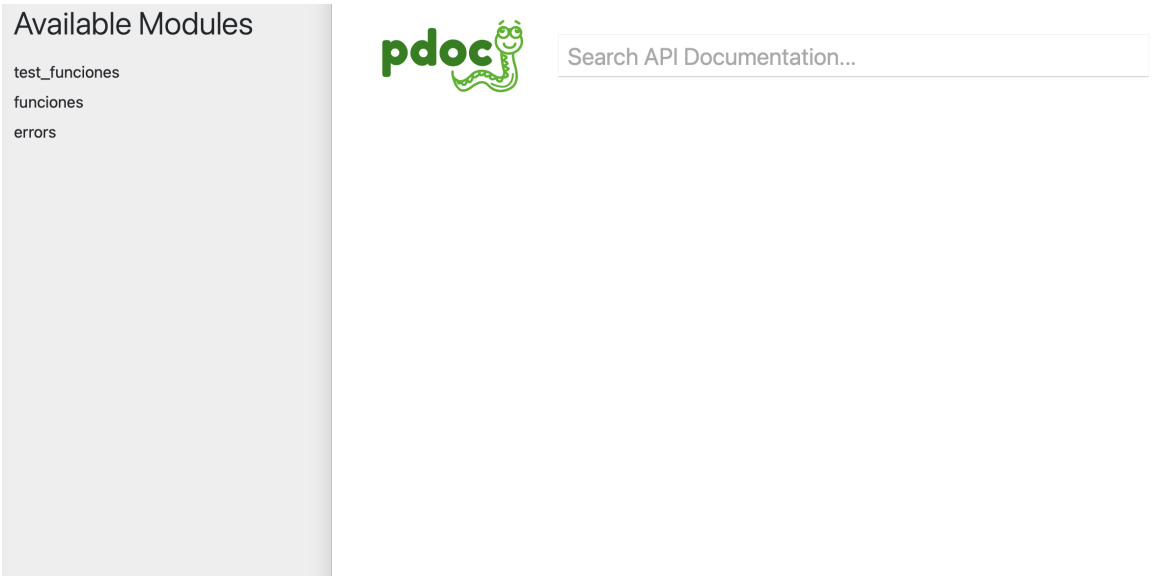

Figure 4: Documentación librería

En caso de que no se desee que `pdoc` abra de forma automática el navegador con la documentación se debe hacer uso del flag `-n` // `--no-browser` seguido de `True` como argumento

Ejemplo

```
pdoc nombre_del_modulo.py -n True
```

> **Nota:** Recordar que una de las características principales de `pdoc` es la recarga automática del navegador cada vez que se editan los módulos a ser docuemntados. Esta característica hace `pdoc` una herramienta bastante útil/versatil para probar y ajustar los docstrings antes de generar los ficheros finales

Creación de ficheros

Una vez finalizadas las pruebas y al estar conforme con el diseño final, podemos generar los ficheros de la documentación a través de `pdoc`

> **Importante**
>
> Antes de generar los ficheros se recomienta crear un directorio llamado `docs` paralelo al directorio donde se encuentran almacenados los módulos a documentar.

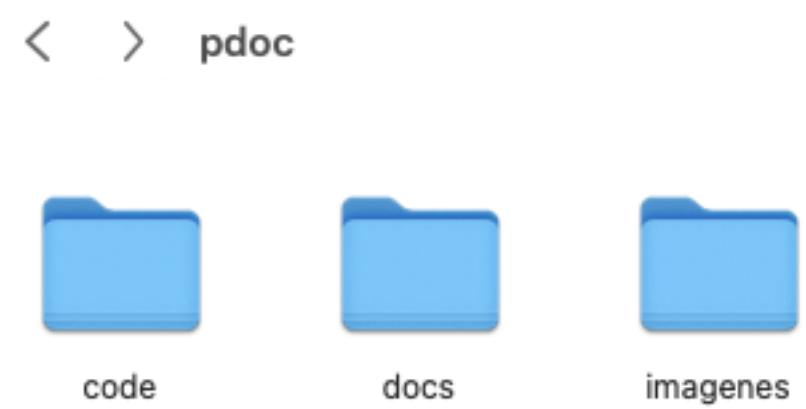

Figure 5: Estructura de directorios

Una vez creado el directorio `docs` procedemos a ejecutar la siguiente instrucción en la terminal.

```
pdoc nombre_del_modulo_1.py nombre_del_modulo_2.py -o ../docs
```

El flag `-o` // `--output-directory` le indica a `pdoc` que se deben generar los ficheros y no iniciar el servidor web. Se debe suministrar la dicción del directiorio donde guardar los ficheros como argumento `../docs`

> **Nota:** La instrucción anterior supone que aún nos encontramos en el directorio que contiene los módulos. En caso de encotrarse la terminal en el directorio `docs` la instrucción sería la siguiente
>
> ```
> pdoc nombre_del_modulo_1.py nombre_del_modulo_2.py -o .
> ```

Una vez generada la documentación podemos buscar el fichero `index.html` y ejecutarlo para poder disfrutar de nuestra documentación de forma “off-line”

Comandos opcionales

- Es posible obtener la documentación de `pdoc` desde la terminal a través del comando

  ```
  pdoc pdoc
  ```

- En caso de customizar el renderizado de la documentación podemos hacer uso del flag `--help`

```
pdoc --help
```

- En nuestro caso partícular se decidió configurar los módulos para evitar que se mostrará los `inherited-members` de las clases en la documentación. Para ello se debe incluir las siguientes líneas de código en cada módulo donde no se desee mostrar dichas variables

```
import pdoc

pdoc.doc.Namespace.inherited_members = {}
```

`inherited-members` habilitado

def test_input_empty_column(self):

Comprobación de la función `funciones.create_df` para inputs de entrada: Columna vacia

Esta prueba unitaria hace uso del fichero `prueba3.csv` para la creación del DataFrame y su posterior prueba

```
>>> path = 'prueba3.csv'
```

Inherited Members

unittest.case.TestCase TestCase, class failureException, longMessage, maxDiff, addTypeEqualityFunc, addCleanup, addClassCleanup, tearDown, setUpClass, tearDownClass, countTestCases, defaultTestResult, shortDescription, id, subTest, run, doCleanups, doClassCleanups, debug, skipTest, fail, assertFalse, assertTrue, assertRaises, assertWarns, assertLogs, assertEqual, assertNotEqual, assertAlmostEqual, assertNotAlmostEqual, assertSequenceEqual, assertListEqual, assertTupleEqual, assertSetEqual, assertIn, assertNotIn, assertIs, assertIsNot, assertDictEqual, assertDictContainsSubset, assertCountEqual, assertMultiLineEqual, assertLess, assertLessEqual, assertGreater, assertGreaterEqual, assertIsNone, assertIsNotNone, assertIsInstance, assertNotIsInstance, assertRaisesRegex, assertWarnsRegex, assertRegex, assertNotRegex, failUnlessRaises, failIf, assertRaisesRegexp, assertRegexpMatches, assertNotRegexpMatches, failUnlessEqual, assertEquals, failIfEqual, assertNotEquals, failUnlessAlmostEqual, assertAlmostEquals, failIfAlmostEqual, assertNotAlmostEquals, failUnless, assert_

Figure 6: Inherited-members habilitado

`inherited-members` deshabilitado

def test_input_empty_column(self):

Comprobación de la función `funciones.create_df` para inputs de entrada: Columna vacia

> Esta prueba unitaria hace uso del fichero `prueba3.csv` para la creación del DataFrame y su posterior prueba

```
>>> path = 'prueba3.csv'
```

class ExpectedFailureTest(unittest.case.TestCase):

Clase que contiene todos las pruebas unitarias donde se espera un fallo durante la evaluación

La clase `ExpectedFailureTest` hereda sus atributos de la clase `TestCase` del paquete `unittest`

> Los métdos definidos dentro de esta clase van acompañados del decorador `@unitttest.expectedFailure`

Figure 7: Inherited-members deshabilitado