Documentación en Python

> "*Code is more often read than written.*"
>
> Guido van Rossum

Y otra cita:

> "*Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing.*"
>
> Dick B.

Comentar vs documentar el código

Before we can go into how to document your Python code, we need to distinguish documenting from commenting.

Commenting your code serves multiple purposes, including:

- Planning and Reviewing:
- Code Description
- Algorithmic Description:
- Tagging: The use of tagging can be used to label specific sections of code where known issues or areas of improvement are located.
  Some examples are: BUG, FIXME, and TODO.

```
## TODO: pendiente añadir feature 123 bla, bla..
```

Documentar el código usando Docstrings

Antes observemos la ayuda que nos proporciona el CLI sobre comandos,clases, etc. usando `help()`:

```
>>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object ....
 |  ....
>>>
```

Podemos ver todo lo relacioando con el objeto string con `dir()`, incluido el atributo `.__doc__`

```
>>> dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__',
...
```

Si accedemos a dicho atributo del objeto vemos su ayuda:

```
>>> print(str.__doc__)
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
...
>>>
```

Es decir, podemos crear o modificar ese atributo de una función propia:

```
>>> def hola(name):
...     print(f"hola {name}. Bienvenido")
...
>>> hola.__doc__ = "Funcion que saluda con nombre"
>>> help(hola)
Help on function hola in module __main__:

hola(name)
    funcion que saluda con nombre
(END)
>>>
```

Este método no sería práctico para documentar. En su lugar es prefible usar comentarios *de una determinada forma*. Observe que el comentario de la función del siguinte ejemplo está justo al inicio de la funciám, tabulado, etc. Si lo definimos de esa forma se le asigna al atributo `.__doc__`

```
>>> def hola(name):
...     """ Funcion que saluda con parametro nombre."""
...     print(f"hola {name}. Bienvenido")
...
>>> help(hola)
Help on function hola in module __main__:

hola(name)
    Funcion que saluda con parametro nombre.
(END)
>>>
```

Vamos a ver entonces que son los Docstrings y cómo utilizarlos.

docstrings

*A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the **doc** special attribute of that object.*

Docstring conventions are described within PEP 257 (*Python Enhancement Proposals*)
https://www.python.org/dev/peps/pep-0257/

docstrings de una línea

Como en ejemplo anterior. Empieza con mayúscula, termina en punto. Sin líneas en blanco ni antes ni después.

docstrings multilínea

Este sería el formato aproximado:

```
"""This is the summary line

This is the further elaboration of the docstring. Within this section,
you can elaborate further on details as appropriate for the situation.
Notice that the summary and the elaboration is separated by a blank new
line.
"""

# Notice the blank line above. Code should continue on this line.
```

Observe este ejemplo tomado de PEP 257:

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
```

```
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """

    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

Por tipos y formatos

Existen ciertas convenciones para usarlos con

- paquetes o módulos
- clases
- scripts

Y existen distintas "convenciones" de utilizarlos

- Google docstrings
- NumPy/SciPy docstrings
- Epytext
- reStructured Text

Generando documentación desde el código

Exite la posibilidad de generar la documentación de un proyecto, modulo, clase, etc a partir de los docstrings del código.

Hay diferentes solucionesy nosotros usaremos una de ellas, el módulo pdoc.

Instalamos dicho modulo pdoc en un entorno virtual:

```
(venv) @tos:~/docstring$ pip install pdoc3
Collecting pdoc3
  Downloading pdoc3-0.10.0-py3-none-any.whl (135 kB)
     |████████████████████████████████| 135 kB 3.6 MB/s
Collecting mako
  Downloading Mako-1.1.6-py2.py3-none-any.whl (75 kB)
     |████████████████████████████████| 75 kB 6.4 MB/s
Collecting markdown>=3.0
  Using cached Markdown-3.3.6-py3-none-any.whl (97 kB)
Collecting MarkupSafe>=0.9.2
  Using cached MarkupSafe-2.0.1-cp38-cp38-manylinux2010_x86_64.whl (30 kB)
Collecting importlib-metadata>=4.4; python_version < "3.10"
  Using cached importlib_metadata-4.8.2-py3-none-any.whl (17 kB)
Collecting zipp>=0.5
  Using cached zipp-3.6.0-py3-none-any.whl (5.3 kB)
Installing collected packages: MarkupSafe, mako, zipp, importlib-metadata,
markdown, pdoc3
Successfully installed MarkupSafe-2.0.1 importlib-metadata-4.8.2 mako-1.1.6
markdown-3.3.6 pdoc3-0.10.0 zipp-3.6.0
```

```
(venv) @tos:~/docstring$pdoc3 --version
pdoc3 0.10.0
(venv) @tos:~/docstring$
```

Si lo probamos con el módulo calculadora muestra la documentación generada en la salida estándar:

```
(venv) @tos:~/docstring$ pdoc calculadora
Module calculadora
==================

Functions
---------


`resta(a, b)`
:   Resta dos números.


`suma(a, b)`
:   Suma dos números.
(venv) @tos:~/docstring$
```

Es mejor generar dicha documentación en formato HTML:

```
(venv) @tos:~/docstring$ pdoc --html calculadora
html/calculadora.html
(venv) @tos:~/docstring$ firefox html/calculadora.html
```

Observe la presentación de la documentación, la posibilidad de ver el propio código, etc.

Tiene a continuación un ejemplo de la defínción de una clase

```
"""
This module deals with all animals. So far, only dogs are implemented.

# TODO

 - Add the rest of the animals.

"""
from __future__ import annotations

class Dog:
    name: str
    """The name of this dog. May contain non-ASCII characters."""
```

```
    friends: list ["Dog"] ## en Python 3.9...
    """Friends this dog has made."""

    def __init__(self, name: str):
        """Make a Dog without any friends (yet)."""
        self.name = name
        self.friends = []

    def bark(self, loud: bool = True):
        """*woof*"""
```

Y cómo se vería la documentación asociada:

Ejercicio: generar documentación

- Crear calculadora.py con funciones suma y resta.
- Documentar siguiendo docstring de google en una y numpydoc en la otra. Documentar los argumentos de las funciones y los valores de retorno

Nota: Puede usar pdoc en modo servidor apra ir viendo los cambios a medida que edita:

```
(venv) @tos:~/bloque_01/docstring$ pdoc3 --http localhost:8080 calculadora
Starting pdoc server on localhost:8080
pdoc server ready at http://localhost:8080
...
```

Extra: Commenting Code via Type Hinting (Python 3.5+)

Es posible añadir información sobre los tipos de los argumentos o el valor de retorno en el propio código. Pero no hay comprobación de tipos:

```
# función sin tipos..
def hello(name):
    return(f"Hello {name}")

#funciones con tipos de parámetros y tipo retorno
def hello_name(name: str) -> str:
    return(f"Hello {name}")

def headline(text: str, align: bool = True) -> str:
    if align:
        return f"{text.title()}\n{'-' * len(text)}"
    else:
        return f" {text.title()} ".center(50, "o")

print(hello("Ana"))
```

```
print(hello_name("Ana"))
print(headline(" lorem ipsum"))
print(headline(" lorem ipsum", False))
print(headline(" lorem ipsum", "testing"))

print(headline(" lorem ipsum", "test"))
print(hello_name(123))
print(hello_name(False))
```

Enlaces

- https://www.python.org/dev/peps/pep-0257/
- https://pdoc.dev/
- https://realpython.com/documenting-python-code/?utm_source=pocket_mylist
- https://numpydoc.readthedocs.io/en/latest/format.html
- https://google.github.io/styleguide/pyguide.html#38-comments-and-docstrings