

Evitar el código espagueti

Recomendaciones al iniciar un proyecto en python

Francisco Gárate Santiago - fgarate@ucm.es

UCM

Master en Ciencias Actuariales y Financieras (2024-2025)



¿Qué es el código espagueti?



Definición wikipedia:

El código espagueti es un término peyorativo para los programas de computación que tienen una estructura de control de flujo compleja e incomprensible. Su nombre deriva del hecho que este tipo de código parece asemejarse a un plato de espaguetis, es decir, un montón de hilos intrincados y anudados.

Código espagueti

- Quien programa, por naturaleza, tenemos tendencia a crear código que resulta confuso y difícil de mantener. Sobre todo, si no somos desarrolladores informáticos.
- Esto es debido a que estamos programados para buscar la solución de más mínimo esfuerzo, pensando en el corto plazo.
- Resultado, un código espagueti: un código desordenado, difícil de leer y mantener debido a la falta de estructura, con muchas dependencias entre las partes del programa.
- A continuación, vamos a ver una serie de consejos y librerías (como kedro, pylint, black) que sin entrar en mayor complejidad ni menor agilidad nos ayudará a mantener un código limpio.

Evitar el código espagueti

Pasos

- Importancia de la **legibilidad**: nombre de variables, comentarios . . .
- **Modularización** del código: crear nuestro paquete de funciones. Git
- Uso de control de **flujo** claro: kedro
- Pruebas y **depuración** (Linter): pylint y black

Legibilidad

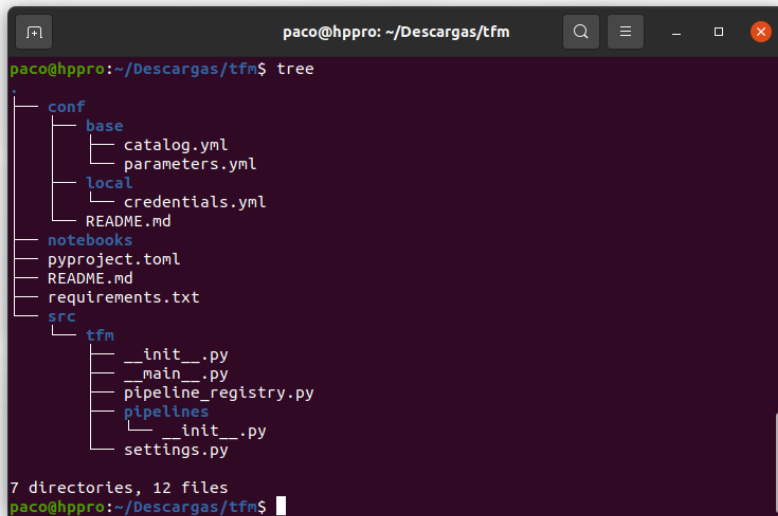
- Estructura el scripts para que se llamen al inicio a todas las librerías, y se eviten establecer variables.
- Añade comentarios usando `#` (en líneas separadas)
- Al comentar no necesario que se explique lo que hace el código, sino el **por qué** lo hace.
- Añade secciones para separar visualmente el código.

¿por qué estructurar tu código?

- **Limpieza:** Escribir código que sea legible y comprensible. Es algo por lo que el tú del futuro te lo agradecerá.
- **Reutilización:** Hace que el trabajo sea más fácil de replicar.
- **Comunidad:** Compartir el código evita duplicar trabajo. Es algo que otras personas te agradecerán.

Estructura de carpetas

A parte de los datos, se tendría que tener separadamente: código, notebooks, requirements.txt

A terminal window with a dark purple background. The title bar shows 'paco@hppro: ~/Descargas/tfm'. The command 'tree' has been executed, displaying a directory tree. The tree shows a root directory with subdirectories 'conf', 'notebooks', and 'src', along with files 'pyproject.toml', 'README.md', and 'requirements.txt'. The 'conf' directory contains 'base' (with 'catalog.yml' and 'parameters.yml'), 'local' (with 'credentials.yml'), and 'README.md'. The 'src' directory contains a 'tfm' subdirectory with '__init__.py', '__main__.py', 'pipeline_registry.py', 'pipelines' (with '__init__.py'), and 'settings.py'. At the bottom, it says '7 directories, 12 files'.

```
paco@hppro: ~/Descargas/tfm$ tree
.
├── conf
│   ├── base
│   │   ├── catalog.yml
│   │   └── parameters.yml
│   ├── local
│   │   └── credentials.yml
│   └── README.md
├── notebooks
├── pyproject.toml
├── README.md
├── requirements.txt
└── src
    └── tfm
        ├── __init__.py
        ├── __main__.py
        ├── pipeline_registry.py
        ├── pipelines
        │   └── __init__.py
        └── settings.py

7 directories, 12 files
paco@hppro:~/Descargas/tfm$
```

Alojamiento de datos

Usar rutas relativas

Usar:

```
df = pd.read_csv("../data/01_raw/DataCAR.csv")
```

En vez de:

```
df = pd.read_csv("c:/Users/paco/OneDrive/Datos/DataCAR.csv")
```


Variables

- Variables. Cuando sean necesarias:
 - Si son muchas: Exporta las variables en un documento aparte
 - Si son pocas: Al inicio del script o en un .py independiente
- Separar código y variables en distintos ficheros te permite poder reutilizar la misma variable en todos los scripts.

Ejemplo:

- Fichero variables.py

```
fval = pd.to_datetime("2024-12-31")
```

- Fichero principal:

```
from variables import fval
```

- El objetivo es que, por ejemplo, cuando tengamos que cambiar la fecha de valoración de nuestro proyecto no tengamos que ir fichero por fichero modificandolo.

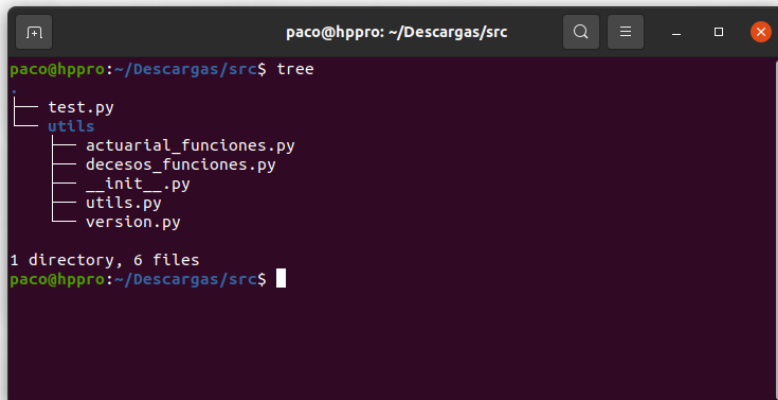
Modularización del código

Exportar tus funciones

- Dividir el código en pequeñas funciones y módulos que realicen tareas específicas.
- La modularización facilita la reutilización del código y lo hace más comprensible y fácil de probar.

Modularización del código

Exportar tus funciones

A terminal window with a dark background and light text. The title bar shows 'paco@hppro: ~/Descargas/src'. The prompt is 'paco@hppro:~/Descargas/src\$'. The command 'tree' has been executed, showing a directory structure with 'test.py' and a subdirectory 'utils'. The 'utils' directory contains 'actuarial_funciones.py', 'decesos_funciones.py', '__init__.py', 'utils.py', and 'version.py'. Below the tree output, it says '1 directory, 6 files'. The prompt is now 'paco@hppro:~/Descargas/src\$' with a cursor.

```
paco@hppro:~/Descargas/src$ tree
.
├── test.py
└── utils
    ├── actuarial_funciones.py
    ├── decesos_funciones.py
    ├── __init__.py
    ├── utils.py
    └── version.py

1 directory, 6 files
paco@hppro:~/Descargas/src$
```

Modularización del código

Exportar tus funciones

Fichero `__init__.py`



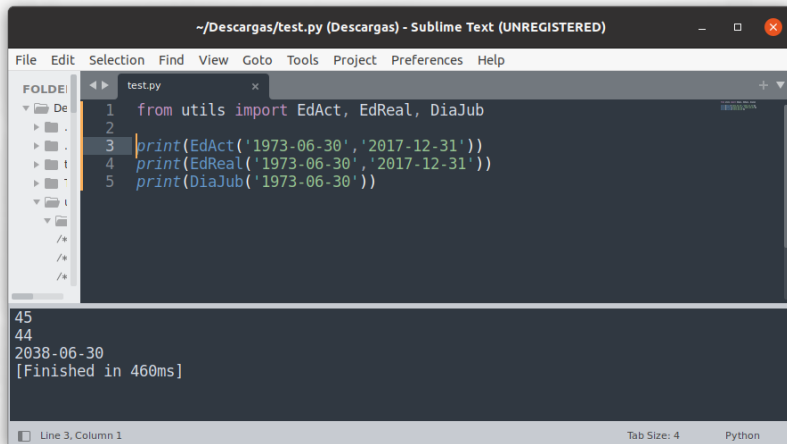
The screenshot shows a code editor window with a dark theme. The title bar indicates the file is `__init__.py` located at `~/Descargas/utills`. The editor contains the following Python code:

```
from .utils import *
from .actuarial_funciones import *
from .version import version
from .decesos_funciones import *
```

Modularización del código

Exportar tus funciones

En este caso, por ejemplo, estas funciones están en el fichero `actuarial_funciones.py`



```
~/Descargas/test.py (Descargas) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDE| test.py x
1 from utils import EdAct, EdReal, DiaJub
2
3 print(EdAct('1973-06-30', '2017-12-31'))
4 print(EdReal('1973-06-30', '2017-12-31'))
5 print(DiaJub('1973-06-30'))

45
44
2038-06-30
[Finished in 460ms]
Line 3, Column 1 Tab Size: 4 Python
```

Modularización del código

Github

- Subir nuestro propio paquete con las funciones que más utilizamos a un repositorio (privado a público) de Github facilita tenerlo siempre disponible y actualizado entre todos nuestros proyectos.
- Ejemplo:

```
git clone https://github.com/franciscogarate/utis_ucm
```

Pruebas y depuración

Revisa tu código

la importancia de probar y depurar el código a medida que se desarrolla. Animar a los estudiantes a escribir pruebas unitarias y a utilizar herramientas de depuración para detectar errores y mantener el código limpio y organizado.

Un **linter** es una herramienta que analiza el código para detectar errores, malas prácticas o incumplimientos de los estándares de estilo.

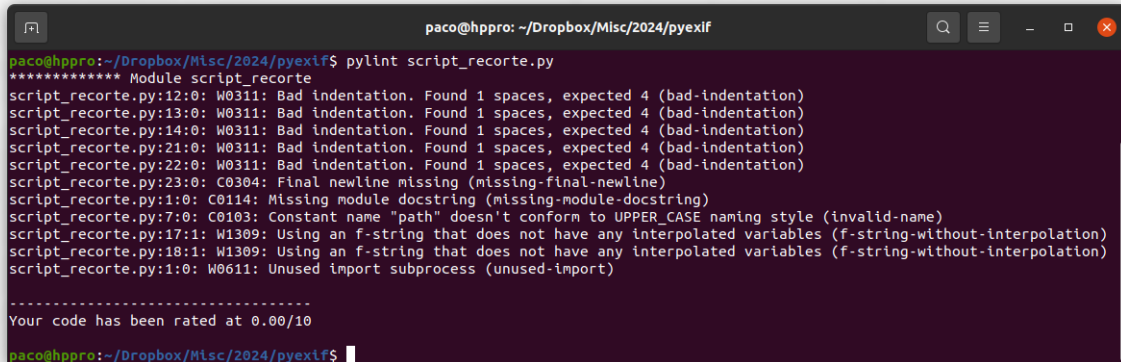
Un **formateador** te ayuda a estructurar tu código, principalmente añadiendo separaciones, tabulaciones, etc...

- Linter para python:
 - **pylint**
 - **Flake8**
- Formateadores:
 - **black**

Pruebas y depuración

pylint

Ejemplo: `pylint script_recorte.py`



```
paco@hppro: ~/Dropbox/Misc/2024/pyexif
paco@hppro:~/Dropbox/Misc/2024/pyexif$ pylint script_recorte.py
***** Module script_recorte
script_recorte.py:12:0: W0311: Bad indentation. Found 1 spaces, expected 4 (bad-indentation)
script_recorte.py:13:0: W0311: Bad indentation. Found 1 spaces, expected 4 (bad-indentation)
script_recorte.py:14:0: W0311: Bad indentation. Found 1 spaces, expected 4 (bad-indentation)
script_recorte.py:21:0: W0311: Bad indentation. Found 1 spaces, expected 4 (bad-indentation)
script_recorte.py:22:0: W0311: Bad indentation. Found 1 spaces, expected 4 (bad-indentation)
script_recorte.py:23:0: C0304: Final newline missing (missing-final-newline)
script_recorte.py:1:0: C0114: Missing module docstring (missing-module-docstring)
script_recorte.py:7:0: C0103: Constant name "path" doesn't conform to UPPER_CASE naming style (invalid-name)
script_recorte.py:17:1: W1309: Using an f-string that does not have any interpolated variables (f-string-without-interpolation)
script_recorte.py:18:1: W1309: Using an f-string that does not have any interpolated variables (f-string-without-interpolation)
script_recorte.py:1:0: W0611: Unused import subprocess (unused-import)

-----
Your code has been rated at 0.00/10

paco@hppro:~/Dropbox/Misc/2024/pyexif$
```


Pruebas y depuración

Black

- Instalación fácil:

```
pip install black
```

- Ejecución como script:

```
black source_file_or_directory
```

- También ejecutar Black como un paquete si como un script no funciona:

```
python -m black source_file_or_directory
```

Pruebas y depuración

Black app: <https://black.vercel.app>

Black v24.10.0 - The uncompromising Python code formatter.

Playground built by José Padilla

```
1 from seven_dwarfs import Grumpy, Happy, Sleepy, Bashful, Sneezzy, Dopey, Doc
2 x = { 'a':37, 'b':42,
3
4 'c':927}
5
6 x = 123456789.123456789E123456789
7
8 if very_long_variable_name is not None and \
9 very_long_variable_name.field > 0 or \
10 very_long_variable_name.is_debug:
11 z = 'hello '+'world'
12 else:
13 world = 'world'
14 a = 'hello {}'.format(world)
15 f = rf'hello {world}'
16 if (this
17 and that): y = 'hello 'world'#FIXME: https://github.com/psf/black/issues/26
18 class Foo ( object ):
19     def f (self ):
20         return 37*-2
21     def g(self, x,y=42):
22         return y
23     def f ( a: List[ int ] ):
24         return 37-a[42-u : y**3]
25 def very_important_function(template: str,*variables,file: os.PathLike,debug:bool=False,):
26     """Applies 'variables' to the 'template' and writes to 'file'."""
27     with open(file, "w") as f:
28         ...
29     # fmt: off
30     custom_formatting = [
31         0, 1, 2,
32         3, 4, 5,
33         6, 7, 8,
34     ]
35     # fmt: on
36     regular_formatting = [
37         0, 1, 2,
38         3, 4, 5,
39         6, 7, 8,
40 ]
```

```
1 from seven_dwarfs import Grumpy, Happy, Sleepy, Bashful, Sneezzy, Dopey, Doc
2
3 x = ("a": 37, "b": 42, "c": 927)
4
5 x = 123456789.123456789E123456789
6
7 if (
8     very_long_variable_name is not None
9     and very_long_variable_name.field > 0
10     or very_long_variable_name.is_debug
11 ):
12     z = "hello " + "world"
13 else:
14     world = "world"
15     a = "hello {}".format(world)
16     f = rf"hello {world}"
17 if this and that:
18     y = "hello " "world" # FIXME: https://github.com/psf/black/issues/26
19
20
21 class Foo(object):
22     def f(self):
23         return 37 * -2
24
25     def g(self, x, y=42):
26         return y
27
28
29 def f(a: List[int]):
30     return 37 - a[42 - u : y**3]
31
32
33 def very_important_function(
34     template: str,
35     *variables,
36     file: os.PathLike,
37     debug: bool = False,
38 ):
39     """Applies 'variables' to the 'template' and writes to 'file'."""
40     with open(file, "w") as f:
41         ...
42
43
44 # fmt: off
45 custom_formatting = [
46     0, 1, 2,
```



Report issue