Manual sucinto de estilo para Python

Una función bien diseñada ha de ser *correcta*, *eficiente* y *mantenible*. Este último adjetivo asume como primera característica la *legibilidad*. Es decir, el diseño de las funciones y módulos ha de tener en cuenta su escritura de forma *legible*, por uno mismo y por los demás miembros de la comunidad Python.

Por lo tanto, la *claridad* es en principio básico. Esto incluye, entre otras cosas, la elección adecuada de identificadores (nemotécnicos) y el uso de expresiones y recursos del lenguaje de forma lo más clara que sea posible, y la documentación del código.

Pero además, un programa, un módulo o un script va a ser leído por otros, de manera que conviene adoptar habitos estándar, no innovar un estilo personal distinto de dichos estándares. Estas convenciones ayudan a cada miembro de la comunidad Python a antender con mayor naturalidad y fluidez las piezas de código de los demás.

El principal estándar actualmente se denomina `PEP8`, y contiene un buen número de convenciones para la documentación del código. Hay muchas guías disponibles en Internet en distintos idiomas y con un grado variable de detalle. He aquí una recomendable en mi opinión por tener una buena relación legibilidad/tamaño:

PEP 8 -- Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/)

En este pequeño documento, damos una pequeña selección de dichas directrices, aceptando que es incompleta, pero útil como una introducción sucinta. En ella he recogido algunas que me han parecido básicas, y lógicamente remito a quien lo lea a completar esta lectura con la de otras guías más completas.

Ojalá te resulte útil.

Identificadores

- Los identificadores deben representar el dato o función representado, no el tipo de datos en el que se almacenan. Es adecuado, por ejemplo, `edad` si es esto lo que representa una variable entera, y no `entero`. Se evitará usar el identificador `l`, por confundirse con un 1. En los índices de los bucles, es corriente usar `i`, `j`, `k`.
- Los identificadores se limitarán a usar caracteres ASCII, es decir, que no se usarán tildes ni eñes en los identificadores.
- Es adecuado y recomendable que un identificador tenga varias palabras. En este caso, hay dos tipos de notaciones:
  - Guiones bajos: `paises_de_europa`

    Esta notación se usa para los identificadores que empiezan con minúscula: nombres de funciones, variables, módulos, parámetros de funciones.
  - Mayúsculas o notación camello: `PaisesDeEuropa`

    Esta notación se usa para los identificadores de constantes y de clases.

In [1]:

```
# Ejemplos:

import math
PI = math.pi
radio = 4.5
area_del_circulo = PI * radio**2
print(area_del_circulo)
```

```
63.61725123519331
```

Escritura del código

- Adoptar un sangrado de cuatro espacios
- Limitar la longitud de las líneas a 79 caracteres
- Cuando se requiera rebasar esta longitud...
  - Se puede poner una barra invertida
- Las relaciones de datos entre paréntesis, corchetes o llaves:
  - Se separan con una coma y un espacio
  - Se pueden desglosar en líneas; en este caso, el sangrado puede ponerse a cuatro espacios y también justificado tras el primer delimitador, y el cierre puede justificarse como los datos o aliearse ya con la línea que inicó el sangrado.

In [2]:

```
# Ejemplos:

for i in range(10):
    for j in range(i):
        print(i, end="")
    print()

print()

for pais, abreviatura in [("España", "SP"), ("Francia", "FR")]:
    print(pais, abreviatura)

print()

for pais, abreviatura in [("España", "SP"), ("Francia", "FR"), \
                          ("Portugal", "PT"), ("Gran Bretaña", "GB")]:
    print(pais, abreviatura)

print()

for pais, abreviatura in [
    ("España", "SP"), ("Francia", "FR"), \
    ("Portugal", "PT"), ("Gran Bretaña", "GB")
]:
    print(pais, abreviatura)
```

```
1
22
333
4444
55555
666666
7777777
88888888
999999999

España SP
Francia FR

España SP
Francia FR
Portugal PT
Gran Bretaña GB

España SP
Francia FR
Portugal PT
Gran Bretaña GB
```

Comentarios

- Los comentarios han de limitarse a usar los caracteres ascii.
- Los comentarios se redactarán de forma escueta, pero serán oraciones completas, empezando con mayúscula y sin cambiar la notación de los identificadores mencionados.
- No se deben incluir comentarios obvios: son molestos y únicamente perturban la lectura del código

- Los comentarios de las funciones y clases se deben organizar en un `docstring`, incluyendo:
  - Una descripción escueta de lo que hace la función (no de cómo lo hace)
  - `Parameters`, cada uno con su tipo y una descripción escueta
  - `Precondition`, cuando sea necesaria
  - `Returns`, incluyendo el tipo y la descripción del objeto devuelto
  - `Example` de funcionamiento, o ejemplos, cuando sea necesario mostrar varias situaciones distintas.

In [3]:

```
# Ejemplo:

def radius_of_circle(area):
    """
    Given the area of a circle, returns its radius

    Parameters:
    -----------
    area: float
        the area of the circle

    Precondition:
    -------------
    area >= 0

    Returns:
    --------
    float
        The radios of the circle

    Example:
    --------
    >>> radius_of_circle(9.0)
    1.692568750643269
    """

    PI = math.pi
    return math.sqrt(area / PI)
```

Una función que no tiene `return` debe documentarlo así:

```
    """
    Returns:
    --------
    NoneType
    """
```

Librerías

- Deben importarse al inicio de un script, una sola vez, no dentro de las funciones ni justo antes de su uso. Es verdad que casi todos los libros, manuales y apuntes las colocan donde se necesitan. Se hade esto para evidenciar su necesidad y aclarar que se ha de haber realizado la importación, pero esto es principalmente a efectos didácticos.
- El orden aconsejado es: primero, las librerías estándar y luego las diseñadas por nosotros mismos, en su caso.

- La importación de las librerías se hará en una por línea:

In [4]:

```
import math
import matplotlib.pyplot as plt
import pandas as pd
from collections import defaultdict
```

En la importación de librerías, debe evitarse el uso de comodines (*wildcard*):

```
from here import *
```

Espacios y otros detalles

- Se dejará un espacio a la izquierda y derecha del operador de asignación.
- Se dejará un espacio tras la coma; no antes.
- No se separará el nombre de una función y el paréntesis de los parámetros
- Cuando intervienen varias operaciones, los espacios pueden ayudar a leer mejor las fórmulas, segun la prioridad de las operaciones

In [5]:

```
x, y = 2.5, 5.0
z = math.sin(math.pi*5)
print(z**4 + 4*x**3)
```

```
62.5
```

Comprobación de estilo online

Se han desarrollado unas cuantas herramientas que comprueban el estilo de un programa.

He aquí una muy sencillita, online, y práctica para empezar:

PEP8 online (http://pep8online.com/)

A manera de ejercicio, te doy un fragmento de código para que compruebes su estilo en esa herramienta y repares los defectillos que te vaya diciendo.

In [6]:

```
def funcion_rara( parametro_primero ,    parametro_segundo,
    parametro_tercero ):

  """
  Devuelve los datos dados, cambiados de orden
  """

  return parametro_tercero ,parametro_segundo , parametro_primero
```

In [7]:

```
funcion_rara(2, 3, 4)
```

Out[7]:

```
(4, 3, 2)
```

Una referencia, entre muchas posibles

- PEP 8 en Español – Guía de estilo para el código Python www.recursospython.com/pep8es.pdf (http://www.recursospython.com/pep8es.pdf)