**PEP – Python Enhancement Proposals**

* PEP0 🡪 Listado de PEPs. <https://peps.python.org/>
* PEP1 🡪 Proceso de ‘gestión’ de PEPs. <https://peps.python.org/pep-0001/>
* PEP20 🡪 El ZEN de Python. <https://peps.python.org/pep-0020/>

import this 🡪 Muestra los 19 aforismos.

>>> import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than \*right\* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

* PEP 8 – Guía de estilos para el código Python
  + <https://peps.python.org/pep-0008/>
  + Existe la guía de estilos de Google: <https://google.github.io/styleguide/pyguide.html>
  + La guía de estilos (PEP8) se puede matizar o ajustar a la empresa o proyecto dando lugar a una guía de estilos propia.
  + ¡Seamos **coherentes**! La guía de estilos que se decida utilizar se utiliza.
  + ¿Cuándo no seguir las pautas de PEP8?
    - Cuando queremos ser compatibles con versiones.
    - Si seguir las pautas influye negativamente en la legibilidad del código.
    - Cuando queremos ser coherentes con el resto del código.
    - Si no hay una buena razón para ser compatible o el código es anterior PEP8.
  + PEP8 compliant checker.
    - **pycodestyle** (llamado en el pasado **pep8**)
    - instalación**: pip install pycodestyle**
    - uso: **pycodestyle ejercicio\_07\_pycodestyle.py**
    - Documentación: <https://pycodestyle.pycqa.org/en/latest/>
    - Más info: <https://github.com/PyCQA/pycodestyle>
  + PEP8 autoformateador.
    - **autopep8**.
    - <https://pypi.org/project/autopep8/>
    - Se puede utilizar como plug-in de VisualStudio Code

Texto

Descripción generada automáticamente

* + - Se puede utilizar desde un terminal:
      * pip install autopep8
      * autopep8 fichero.py 🡪 Muestra la versión ‘corregida’ por consola.
      * autopep8 --in-place --aggressive --aggressive fichero.py 🡪 Modifica el fichero de entrada.
  + PEP8 – Sangría (indentación)
    - https://peps.python.org/pep-0008/#indentation
    - A nivel sintáctico pueden ser tabulaciones o espacios en un número indeterminado, pero debe haber consistencia a nivel de bloque (si usan tabuladores, todo el bloque estará sangrado con tabuladores; la misma regla si se utilizan espacios en blanco) (si se usan espacios en blanco debe de utilizar el número a nivel de bloque, pero puede haber diferente número en diferentes bloques).
    - Desde el punto de vista de la consistencia con PEP8 deben ser **4 ESPACIOS EN BLANCO**.
  + PEP8 – Líneas de continuación
    - Cuando una línea excede del tamaño máximo (79 caracteres) ¿cómo la partimos?

def funcion(a, b, c, d, e, f):

pass

lista = [1, 2, 3, 4, 5, 6]

funcion(10, 11, 12, 13, 14, 15)

#MAL

lista = [1, 2, 3,

4, 5, 6

]

funcion(10, 11, 12

13, 14, 15)

**#BIEN**

**lista = [**

**1, 2, 3,**

**4, 5, 6**

**]**

**funcion(10, 11, 12,**

**13, 14, 15)**

**lista = [**

**1, 2, 3,**

**4, 5, 6**

**]**

**funcion(10, 11, 12,**

**13, 14, 15)**

**funcion(**

**10, 11, 12,**

**13, 14, 15)**

**funcion(**

**10, 11, 12,**

**13, 14, 15**

**)**

* + PEP8 – Líneas en blanco
    - Dos líneas en blanco para rodear definiciones de funciones y de clases.
    - Dentro de las clases, una única línea en blanco para separar los métodos, excepto el primer método que podría ir pegado a la declaración de la clase si no hay atributos de clases.

class FiguraGeometrica:

sistema = decimal

def \_\_init\_\_(self):

pass

def dibujar\_figura(self):

pass

def colorear\_figura(self):

pass

* + - Se pueden utilizar una o dos líneas en blanco para separar bloques lógicos de código (con moderación).
  + PEP8 – Codificaciones de caracteres.
    - Python3 🡪 UTF-8
    - Python2 🡪 ASCII
    - Elementos propios de la biblioteca de Python:
      * ASCII
      * Utilizan palabras en inglés
    - Nuestros elementos:
      * Es recomendable utilizar ASCII y ¿utilizar palabras en inglés?
  + PEP8 – Imports
    - Siempre al principio del módulo (inmediatamente después del docstring del éste).
      * Primero los import de la biblioteca estándar
      * Segundo los import de bibliotecas de terceros
      * Por último, los import de módulos propios
    - Incluir una línea de separación entre cada uno de los tres bloques anteriores.
    - No incluir varios en una única línea. Mejor tener tantas líneas como import.

# Mal

import sys, os

# Bien

import smtplib

import socketserver

* + - Sí se puede incluir en una línea la importación de varios elementos de un módulo.

# Bien

from random import gauss, lognormvariate

* + - Utilizar en la medida de lo posible importaciones absolutas.

# Imports absolutos

from instrumentos.materiales.materials import Material

* + - **NO** se recomienda el uso de comodines

from crypt import \*

* + PEP8 - Comillas en las cadenas.
    - Se pueden utilizar comillas o dobles, pero se debe ser consistente.
    - Se recomienda no escapar las comillas 🡪 Intercalar.

# (MAL) Escapar las comillas

cadena\_con\_cadena = "Esto es una \"cadena\""

# (BIEN) Intercalar comillas simples y dobles

cadena\_con\_cadena = "Esto es una 'cadena'"

* + - Si se utilizan comillas triples, se recomienda utilizar comillas dobles (por mantener la consistencia con PEP257).
  + PEP8 – Espacios en blanco en expresiones y declaraciones.
    - No abusar de los espacios en blanco.
    - No utilizar espacios en blanco inmediatamente después de un paréntesis, corchete o llave.
    - No utilizar espacios en blanco inmediatamente antes de dos puntos (:), punto y coma (;) o coma (,)
    - Sobre el slice:
      * Nota: creo que la interpretación de esta norma debe llevar a la eliminación de los espacios en los slices.
      * *However, in a slice the colon acts like a binary operator, and should have equal amounts on either side (treating it as the operator with the lowest priority). In an extended slice, both colons must have the same amount of spacing applied. Exception: when a slice parameter is omitted, the space is omitted*
    - Espacios en blanco en expresiones y declaraciones.
      * No utilizar más de un espacio antes y después de los operadores.
      * Cuando se utilizan operadores de distinta prioridad, se permite no incluir los espacios en los de la más alta. Los paréntesis determinan la prioridad.

# Bien

x = (5 - 8) \* (3 + 2)

x = 5 \* 8 + 3

# Bien (a discreción)

**x = (5-8) \* (2-1)**

**x = 5\*8 + 3**

* + - * Cuando el operador = se utilice para indicar un valor por defecto en una función/método, no se deben de separar con espacios.
  + PEP8 – Trailing commas (comas finales.
    - No utilizar espacios en blanco después de las comas finales.
  + PEP8 – Comentarios
    - Los comentarios deben ser oraciones completas, con la primera letra en mayúsculas y terminadas en punto.

# Esto es un comentario correcto.

# esto no es un comentario bien construido

* + - No deben exceder de 72 caracteres.
    - Escribir los comentarios en inglés.
    - Los comentarios de bloque o de múltiples líneas, se pueden distribuir en párrafos.
      * Cada línea va precedida de # y espacio.
      * Los párrafos se separan por líneas en blanco (una o dos).
      * Deben hacer referencia al código que le sigue.
      * Deben estar sangrados al mismo nivel que el código que describen.

# Muestra por pantalla el texto 'Hola mundo'.

#

# Si recibe un nombre, lo muestra también.

# Es un método muy sencillo.

def saludar(nombre):

pass

class Club:

# Muestra por pantalla el texto 'Hola mundo'.

#

# Si recibe un nombre, lo muestra también.

# Es un método muy sencillo.

def calcular(self):

pass

* + - Comentarios *inline* (en línea).

edad = 18 # Indica la edad mínima de acceso al club.

* + - Documentation strings (contemplado en la PEP257)
      * Orientado a documentar el código para facilitar su uso.
      * Se utilizan las triples comillas dobles como elemento de delimitación inicial y final: “”” documentación “””
  + PEP8 – Convenciones de nomenclatura
    - **No** todos los elementos de la biblioteca de Python son consistentes con la convención.
    - Recomendación 🡪 Todo lo nuevo debe ser consistente con PEP8.
      * Excepción: modificación de algo existente que no lo es.
    - Convenciones de nombres:
      * a 🡪 Una letra minúscula
      * A 🡪 Una letra mayúscula
      * elnombredelavariable 🡪 Todo en minúsculas.
      * **el\_nombre\_de\_la\_variable 🡪 snake\_case**
      * ELNOMBREDELAVARIABLE 🡪 Todo en mayúsculas
      * **EL\_NOMBRE\_DE\_VARIABLE 🡪 SNAKE\_CASE**
      * elNombreDeLaVariable 🡪 Camel-case
      * ElNombreDeLaVariable 🡪 Camel-case
      * El\_Nombre\_De\_La\_Variable 🡪 Se considera ‘feo’
      * **\_el\_nombre\_de\_la\_variable 🡪 (snake\_case) Indica ‘uso interno’.** Ante una importación de un módulo que incluye este tipo de identificadores, con comodín no se importa.
      * **\_\_el\_nombre\_de\_la\_variable 🡪 (snake\_case)** El doble guion baja inicial ‘oculta’ la variable
      * **el\_nombre\_de\_la\_variable\_ 🡪** El guion bajo final proporciona es diferenciar nuestro identificador de identificadores existentes en el lenguaje.
      * \_\_el\_nombre\_de\_variable\_\_ 🡪 Indica que es un método o atributos mágico.
    - No utilizar nombre de variable (cualquier elemento) que tengan una sola letra especialmente las letras ele minúscula (l) ni la o mayúscula (O).
    - Identificadores de variables, funciones y métodos:
      * Deben contener letras minúsculas y separar las palabras por guiones bajos (snake-case)
    - Identificadores de clase:
      * Camel-case comenzando por letra mayúscula.
    - Constantes:
      * Snake-case en mayúsculas: HORA\_LIMITE
    - Módulos:
      * Palabras en minúsculas separadas por guiones bajos (snake-case)
    - Paquetes:
      * Palabras en minúscula sin separador.
    - Excepciones:
      * Mismas reglas que las clases.
  + PEP8 – Recomendaciones
    - Utilizar comparaciones con el objeto **None** y con **is** ó **is not**, en lugar de utilizar los operadores **==** o **!=**
      * **Ejemplo correcto: if x is None**
      * Ejemplo incorrecto: if x == None
    - Utilizar **is** o **is not** en lugar de los operadores **==** ó **!=** para comprobar si algo es **True** o **False**.
      * **Ejemplo correcto: if valor is True**
      * Ejemplo incorrecto: if valor == True
      * **Ejemplo MÁS correcto: if valor**
    - Utilicemos el operador **is not** en lugar de **not … is**
      * **Ejemplo correcto: if x is not None**
      * Ejemplo incorrecto: if not x is None
    - En el caso de la captura de excepciones, no hacer sólo except, sino capturar la excepción indicando su nombre:
      * Mal: except:
      * **Bien: except ValueError**:
    - Utilizar métodos startwidth y endswith de la clase string en lugar de el slicing para determinar cómo empieza o termina una cadena.
      * **Ejemplo correcto: if nombre.startswith(‘Ped’)**
      * Ejemplo incorrecto: if nombre[:3]==’Ped’
* PEP257. Docstring.
  + Delimitado por triples comillas dobles.
  + Pueden ser de una única línea o de varias.
  + Si hay varias líneas, el sangrado debe ser homogéneo.
  + Deben ir inmediatamente después de aquello que documentan.
  + Aplicado a clases, métodos, funciones, se convierten en el atributo \_\_doc\_\_
  + Los editores lo van a utilizar como ayuda ‘contextual’.
  + Formar parte de la ayuda proporcionada a través de la llamada a la función **help(***nombre\_del\_elemento*).
  + Utilizar ‘obligatoriamente’ en métodos públicos (además de funciones, módulos y clases).
  + ¿Utilizar en los métodos privados? Decisión del programador o del proyecto.
  + No poner líneas en blanco antes ni después.
* PEP484. Type hints (sugerencias de tipo).
  + Proporciona ayuda a la hora de informar de los tipos esperados/utilizados/retornados por los componentes de software (funciones o métodos)

def calcular\_area(ancho: int, alto: int) -> int:

"""Calcula el área de un rectángulo

En caso de que los dos lados sean iguales

lanzará un error y escribirá un log.

"""

area = ancho \* alto

return area

RESULTADO = calcular\_area(ancho=3, alto=10)