



PYTHON

Best Practice and Standarization
PEP 8 - Style Guide for Python Code

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- PEP 8 es un documento que proporciona convenciones de codificación (guía de estilo de código) para el código Python.
- PEP 8 se considera uno de los PEP más importantes y una lectura obligada para todos los programadores profesionales de Python, ya que ayuda a que el código sea más consistente, más legible y más eficiente.
- Aunque algunos proyectos de programación pueden adoptar sus propias pautas de estilo (en cuyo caso, dichas pautas específicas del proyecto pueden ser preferibles a las convenciones proporcionadas por PEP 8, especialmente en caso de conflictos o problemas de compatibilidad con versiones anteriores), las mejores prácticas de PEP 8 siguen siendo una lectura muy recomendable, ya que lo ayudan a comprender mejor la filosofía detrás de Python y convertirse en un programador más consciente y competente.
- PEP 8 sigue evolucionando a medida que se identifican e incluyen nuevas convenciones adicionales y, al mismo tiempo, se identifican algunas convenciones antiguas como obsoletas y se desaconseja su seguimiento.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- La coherencia es un factor crucial que determina la legibilidad del código. Por otro lado, la incoherencia con PEP 8 puede ser a veces una mejor opción. Si las guías de estilo no son aplicables a su proyecto, puede ser mejor ignorarlas y decidir por sí mismo qué es lo mejor. Como dice PEP 8:
 - Una guía de estilo se trata de coherencia. La coherencia con esta guía de estilo es importante. La coherencia dentro de un proyecto es más importante. La coherencia dentro de un módulo o función es lo más importante. [...] Sin embargo, sepa cuándo ser incoherente [...]. En caso de duda, utilice su mejor criterio.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- ¿Cuándo debería ignorar algunas pautas específicas de PEP 8 (o al menos considerar hacerlo)?
 - Si seguirlas significará romper la compatibilidad con versiones anteriores.
 - Si seguirlas tendrá un efecto negativo en la legibilidad del código.
 - Si seguirlas causará inconsistencia con el resto del código. (Sin embargo, esta puede ser una buena oportunidad para reescribir el código y hacerlo compatible con PEP 8).
 - Si no hay una buena razón para hacer que el código sea compatible con PEP 8, o el código es anterior a PEP 8.
- PEP 8 tiene como objetivo mejorar la legibilidad del código y "hacerlo consistente en todo el espectro del código Python". Por lo tanto, mantener su código Python compatible con PEP 8 es una buena idea, pero nunca debe adherirse ciegamente a estas recomendaciones. Siempre debe usar su mejor criterio.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

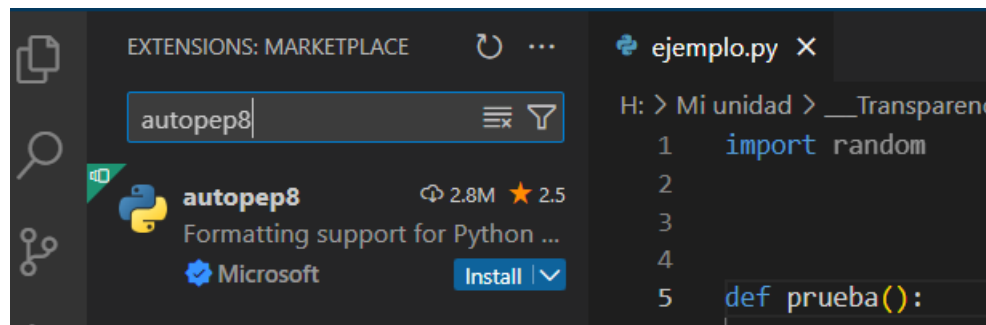
■ **PEP 8 compliant checkers**

- pycodestyle (antes llamado pep8, pero se cambió el nombre para evitar confusiones): comprobador de la guía de estilo de Python; le permite verificar su código Python para verificar su conformidad con las convenciones de estilo de PEP 8. Puede instalar la herramienta con el siguiente comando en la terminal:
 - pip install pycodestyle
- Puede ejecutarlo en un archivo o archivos para obtener información sobre la no conformidad (e indicar errores en el código fuente y su frecuencia).
 - Más información: <https://github.com/PyCQA/pycodestyle>
 - Documentación: <https://pycodestyle.pycqa.org/en/latest/>

PEP 8 - STYLE GUIDE FOR PYTHON CODE

■ **autopep8**

- para formatear automáticamente tu código Python de acuerdo con las pautas de PEP 8. Para poder usarlo, necesitas tener instalado pycodestyle en tu máquina para indicar las partes del código que requieren correcciones de formato.
- Más información: <https://pypi.org/project/autopep8/>



PEP 8 - STYLE GUIDE FOR PYTHON CODE

■ Recomendaciones para el diseño del código

■ Sangría

- El nivel de sangría, entendido como el espacio en blanco inicial (es decir, espacios y tabulaciones) al comienzo de cada línea lógica, se utiliza para agrupar declaraciones.
- Al escribir código en Python, debes recordar seguir estas dos reglas simples:
 - **Usar cuatro espacios por nivel de sangría y;**
 - **Usar espacios en lugar de tabulaciones.**
- Sin embargo, puedes usar tabulaciones cuando desees mantener la coherencia con el código que ya ha sido sangrado con tabulaciones (si no es posible o eficiente hacerlo compatible con PEP 8).
- Nota: Mezclar tabulaciones y espacios para la sangría no está permitido en Python 3. Esto generará una excepción **TabError**:
“TabError: uso inconsistente de tabulaciones y espacios en la sangría”.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Sangría**

Examples:



```
1 # Bad:
2
3 def my_fun_one(x, y):
4     return x * y
5
6 def my_fun_two(a, b):
7     return a + b
8
```



```
1 # Good:
2
3 def my_function(x, y):
4     return x * y
5
```


PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Líneas de continuación**

- Las líneas de continuación (es decir, líneas lógicas de código que desea dividir porque son demasiado largas o porque desea mejorar la legibilidad) están permitidas si se utilizan paréntesis/corchetes/llaves.
 - <https://peps.python.org/pep-0008/#indentation>

PEP 8 - STYLE GUIDE FOR PYTHON CODE


- **Recomendaciones para el diseño del código**
 - **Líneas de continuación**



```
1 # Bad:
2
3 my_list_one = [1, 2, 3,
4               4, 5, 6
5               ]
6
7 a = my_function_name(a, b, c,
8                     d, e, f)
9
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Líneas de continuación**



```
1 # Good:
2
3 my_list_one = [
4     1, 2, 3,
5     4, 5, 6,
6 ]
7
8 a = my_function_name(a, b, c,
9                      |   |   |   |   d, e, f)
10
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Líneas de continuación**



```
1 # Good:
2
3 my_list_two = [
4     1, 2, 3,
5     4, 5, 6,
6 ]
7
8
9 def my_fun(
10     |     a, b, c,
11     |     d, e, f):
12     return (a + b + c) * (d + e + f)
13
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Salto de línea y operadores**

- Si bien en Python está permitido dividir líneas de código antes o después de operadores binarios (siempre que lo haga de manera consistente y que esta convención se haya utilizado en su código anteriormente), se recomienda que siga las sugerencias de estilo de **Donald Knuth** y divida antes de los operadores binarios, ya que esto da como resultado un código más legible y agradable a la vista.

Donald Ervin Knuth (Milwaukee, Wisconsin; 10 de enero de 1938) es un reconocido experto en ciencias de la computación estadounidense y matemático, famoso por su fructífera investigación dentro del análisis de algoritmos y compiladores (Wikipedia)

<https://medium.com/@mimahmetavcil/the-art-of-computer-programming-by-donald-e-knuth-c97ed454ddd6>

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Salto de línea y operadores**

Example:



```
1 # Recommended
2
3 total_fruits = (apples
4                 + pears
5                 + grapes
6                 - (black currants - red currants)
7                 - bananas
8                 + oranges)
9
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Líneas en blanco.**
 - Las líneas en blanco, llamadas espacios verticales, mejoran la legibilidad de su código.
 - Permiten que la persona que lee su código vea la división del código en secciones, lo ayudan a comprender mejor la relación entre las secciones y a captar la lógica de bloques de código dados con mayor facilidad.
 - De la misma manera, usar demasiadas líneas en blanco en su código hará que parezca escaso y más difícil de seguir, por lo que siempre debe tener cuidado de no usarlas en exceso.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Líneas en blanco. Recomendaciones de PEP8.**
 - **Dos líneas en blanco** para rodear las definiciones de funciones y clases de nivel superior:



```
1 class ClassOne:
2     pass
3
4
5 class ClassTwo:
6     pass
7
8
9 def my_top_level_function():
10     return None
11
```


PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Líneas en blanco. Recomendaciones de PEP8.**
 - **Una sola línea en blanco** para rodear las definiciones de métodos dentro de una clase:



```
1 class MyClass:
2     def method_one(self):
3         return None
4
5     def method_two(self):
6         return None
7
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Líneas en blanco. Recomendaciones de PEP8.**
 - **Líneas en blanco** en funciones para indicar secciones lógicas (con moderación):

✓

```
1 def calculate_average():
2     how_many_numbers = int(input("How many numbers? "))
3
4     if how_many_numbers > 0:
5         sum_numbers = 0
6         for i in range(0, how_many_numbers):
7             number = float(input("Enter a number: "))
8             sum_numbers += number
9
10        average = 0
11        average = sum_numbers / how_many_numbers
12
13        return average
14    else:
15        return "Nothing happens."
16
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Codificaciones predeterminadas**

- Se recomienda utilizar las codificaciones predeterminadas de Python (Python 3: UTF-8, Python 2: ASCII). No se recomiendan las codificaciones que no sean las predeterminadas y solo se deben utilizar con fines de prueba o en situaciones en las que los comentarios o las cadenas de documentación utilicen un nombre (por ejemplo, el nombre de un autor) que contenga un carácter que no sea ASCII.
 - La PEP 8 establece que “todos los identificadores de la biblioteca estándar de Python DEBEN utilizar identificadores que solo sean ASCII y DEBEN utilizar palabras en inglés siempre que sea posible”.
 - Nota: Consulte PEP 3131 (Compatibilidad con identificadores no ASCII) para obtener más información sobre la justificación, así como las ventajas y desventajas del uso de identificadores no ASCII.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Imports**

- Siempre debe colocar las importaciones al comienzo de su script, entre los comentarios/docstrings del módulo y las variables globales y constantes del módulo, respetando el siguiente orden:
 1. Importaciones de bibliotecas estándar;
 2. Importaciones de terceros relacionadas;
 3. Importaciones específicas de bibliotecas/aplicaciones locales.
 - Asegúrese de insertar una línea en blanco para separar cada uno de los grupos de importaciones anteriores.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Imports**

- PEP 8 recomienda que sus importaciones se realicen en líneas separadas, en lugar de concentrarlas en una sola línea (hay excepciones, ver siguiente página):



```
1 # Bad:
2
3 import sys, os
4
```



```
1 # Good:
2
3 import os
4 import sys
5
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Imports**

- Es correcto realizar una importación de una línea utilizando la sintaxis `from... import...`:



```
1 from subprocess import Popen, PIPE
```

```
2
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Imports**

- Si es posible, utilice importaciones absolutas (es decir, importaciones que utilicen rutas absolutas separadas por puntos). Por ejemplo:



```
import animals.mammals.dogs.puppies
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Imports**

- No se deben (y en realidad no se puede) utilizar importaciones relativas implícitas, ya que ya no están presentes en Python 3.
 - Se deben evitar utilizar importaciones con comodines, ya que inhiben la legibilidad del código y pueden interferir con algunos de los nombres ya presentes en el espacio de nombres.



```
from animals import *
```


PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Comillas de cadena**

- Python nos permite usar cadenas entre comillas simples (p. ej., 'una cadena') y entre comillas dobles (p. ej., "una cadena"). Son lo mismo, y no hay ninguna recomendación especial en PEP que te diga qué estilo deberías adoptar al escribir tu código. Nuevamente, la regla más importante es: sé consistente con tu elección.
 - Sin embargo, para mejorar la legibilidad, PEP 8 recomienda que intentes evitar el uso de barras invertidas (caracteres de escape) en las cadenas. Esto significa que:
 - si tu cadena contiene caracteres entre comillas simples, se recomienda que uses cadenas entre comillas dobles;
 - si tu cadena contiene caracteres entre comillas dobles, se recomienda que uses cadenas entre comillas simples.
 - En el caso de cadenas entre comillas triples, PEP 8 recomienda que siempre uses caracteres entre comillas dobles para mantener la coherencia con la convención de cadenas de documentación detallada en PEP 257.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Espacios en blanco en expresiones y declaraciones**
 - PEP 8 contiene una sección extensa que muestra ejemplos de usos correctos e incorrectos de los espacios en blanco en el código. En general, debe evitar usar demasiados espacios en blanco, ya que dificultan el seguimiento del código.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Espacios en blanco en expresiones y declaraciones**
 - Por ejemplo, no use demasiados espacios en blanco inmediatamente dentro de paréntesis, corchetes o llaves, o inmediatamente antes de una coma, punto y coma o dos puntos:



```
1 # Bad:
2
3 my_list = ( dog[ 2 ] , 5 , { "year": 1980 } , "string" )
4 if 5 in my_list : print( "Hello!" ) ; print( "Goodbye!" )
5
```



```
1 # Good:
2
3 my_list = (dog[2], 5, {"year": 1980}, "string")
4 if 5 in my_list: print("Hello!"); print("Goodbye!")
5
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Espacios en blanco en expresiones y declaraciones**

- En el caso de un 'slice', los dos puntos deben tener la misma cantidad de espacio en ambos lados (debe actuar como un operador binario) a menos que se omita un parámetro de porción, en cuyo caso también se debe omitir el espacio.:



```
1 # Bad:
2
3 bread[0 : 3], roll[1: 3 :5], bun[3: 5:], donut[ 1: :5 ]
4
```



```
1 # Good:
2
3 bread[0:3], roll[1:3:5], bun[3:5:], donut[1::5]
4
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Espacios en blanco en expresiones y declaraciones**
 - No utilice más de un espacio antes y después de los operadores.



```
1 # Bad:
2
3 a      = 1
4 b      = a      + 2
5 my_string = 'string' * 2
6
```



```
1 # Good:
2
3 a = 1
4 b = a + 2
5 my_string = 'string' * 2
6
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Espacios en blanco en expresiones y declaraciones**

- Rodee los operadores binarios con un solo espacio en ambos lados. Sin embargo, si en su código hay operadores que tienen diferentes prioridades, puede considerar agregar espacios solo alrededor de los operadores de menor prioridad.



```
1 # Bad:
2
3 x=x+3
4 x -=1
5
6 x = x * 2 - 1
7 x = (x - 1) * (x + 2)
8
```



```
1 # Good:
2
3 x = x + 3
4 x -= 1
5
6 x = x*2 - 1 # Use your own judgement.
7 x = (x-1) * (x+2) # Use your own judgement.
8
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Espacios en blanco en expresiones y declaraciones**
 - No rodee el operador = con espacios si se usa para indicar un argumento de palabra clave/valor predeterminado.



```
# Bad:

def my_function(x, y = 2):
    return x * y
```



```
# Good:

def my_function(x, y=2):
    return x * y
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Trailing commas (comas finales)**

- Nuevamente, no use espacios en blanco excesivos:
 - después de una coma final seguida de un paréntesis de cierre, o
 - inmediatamente antes de un paréntesis de apertura que marca el comienzo de la lista de argumentos de una invocación de función, o
 - inmediatamente antes de un paréntesis de apertura que marca el comienzo de la indexación/segmentación.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Trailing commas (comas finales)**



```
1 # Bad:
2
3 my_tuple = (0, 1, 2, )
4 my_function (5)
5 my_dictionary ['key'] = my_list [index]
6
```



```
1 # Good:
2
3 my_tuple = (0, 1, 2,)
4 my_function(5)
5 my_dictionary['key'] = my_list[index]
6
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Recomendaciones para el uso de comentarios**
 - Los comentarios tienen como objetivo mejorar la legibilidad del código sin afectar el resultado del programa. Los buenos programadores documentan su código y explican los fragmentos de código más complejos, de modo que la persona que lea el código comprenda correctamente lo que sucede en el programa. Debe utilizar los comentarios con prudencia y, siempre que sea posible, escribir código que se comente por sí mismo (por ejemplo, dé nombres propios a las variables, funciones y elementos del código).

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Recomendaciones para el uso de comentarios**
 - Hay algunas reglas que debe seguir al dejar comentarios en el código:
 - Escriba comentarios que no contradigan el código ni confundan al lector. Son mucho peores que no incluir ningún comentario.
 - Actualice sus comentarios cuando se actualice su programa.
 - Escriba los comentarios como **oraciones completas** (escriba con mayúscula la primera palabra si no es un identificador y finalice la oración con un punto).

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Recomendaciones para el uso de comentarios**
 - Ejemplo de comentarios como **oración completa**.

```
1  # Program that calculates body mass index (BMI).
2
3  height = float(input("Your height (in meters): "))
4  weight = float(input("Your weight (in kilograms): "))
5  bmi = round(weight / (height*height), 2)
6
7  print("Your BMI: {}".format(bmi))
8
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Recomendaciones para el uso de comentarios**
 - Los comentarios en bloque generalmente consisten en uno o más párrafos formados por oraciones completas, y cada oración termina en un punto.
 - Debes utilizar uno o dos espacios después del punto de final de oración en comentarios de varias oraciones, excepto después de la oración final.
 - Escriba los comentarios en inglés (a menos que esté 120% seguro de que el código nunca será leído por personas que no hablen su idioma).
 - Los comentarios no deben tener más de 72 caracteres por línea.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Bloques de comentarios**

- Los comentarios en bloque suelen ser más largos y se deben utilizar para explicar secciones de código en lugar de líneas específicas. Permiten dejar información para el lector en varias líneas (y varias oraciones). Por lo general, los comentarios en bloque:
 - deben hacer referencia al código que les sigue;
 - deben tener una sangría al mismo nivel que el código que describen.
 - Al escribir comentarios en bloque, comience cada línea con # seguido de un solo espacio y separe los párrafos con una línea que contenga solo el símbolo #.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Bloques de comentarios**

```
1 def calculate_product():
2     # Calculate the average of three numbers obtained from the user. Then
3     # multiply the result by 4.17, and assign it to the product variable.
4     #
5     # Return the value passed to the product variable and use it
6     # for the subsequent x to y calculations to speed up the process.
7     sum_numbers = 0
8
9     for number in range(0, 3):
10         number = float(input("Enter a number: "))
11         sum_numbers += number
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Comentarios en línea**

- Los comentarios en línea son comentarios que se escriben en la misma línea que sus declaraciones.
 - Deben referirse a una sola línea de código o una sola declaración o brindar una explicación más detallada de la misma.
 - No debe abusar de ellos.
 - En general, los comentarios en línea deben:
 - estar separados por dos (o más) espacios de la declaración a la que se refieren;
 - usarse con moderación.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Comentarios en línea**

Pueden ayudar a recordar rápidamente lo que hace una línea de código en particular o ser útiles cuando los lea alguien que no esté familiarizado con su código.



```
1 counter = 0      # Initialize the counter.  
2
```

No utilices comentarios en línea (ni ningún otro tipo de comentario) para explicar cosas obvias o innecesarias.



```
1 a += 1          # Increment a.  
2
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Comentarios en línea**

Intente siempre hacer que su código se comente por sí solo en lugar de agregar comentarios, incluso si parecen sensatos o necesarios..



```
1 # Bad:
2
3 a = 'Adam' # User's first name.
4
```



```
1 # Good:
2
3 user_first_name = 'Adam'
4
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Documentation strings**

- Las cadenas de documentación, o **docstrings** como se las suele llamar, te permiten proporcionar descripciones y explicaciones para todos los módulos, archivos, funciones, clases y métodos públicos que utilizas en tu código. Deberías utilizarlas en este contexto.
 - Tienen relación con la PEP 257.
 - Son un tipo de comentario que comienza y termina con tres comillas dobles: `"""`.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Documentation strings**

```
1 # A multi-line docstring:
2
3 def fun(x, y):
4     """Convert x and y to strings,
5     and return a list of strings.
6     """
7     ...
8
```

```
1 # A single-line docstring:
2
3 def fun(x):
4     """Return the square root of x."""
5     ...
6
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Convenciones de nomenclatura – Introducción**
 - Dar nombres adecuados y evitar los inapropiados aumentará sin duda la legibilidad del código y le ahorrará a usted (y a otros programadores que lean su código) mucho tiempo y esfuerzo.
 - Las convenciones de nombres de Python no son completamente consistentes en toda la biblioteca de Python. Sin embargo, se recomienda que los nuevos módulos y paquetes se escriban de conformidad con las recomendaciones de nombres de PEP 8 (a menos que una biblioteca existente siga un estilo diferente, en cuyo caso la coherencia interna es la solución preferida).

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Estilos (convenciones) de nombres**

- Existen muchos estilos de nombres diferentes que se utilizan en programación, por ejemplo:
 - a: una sola letra minúscula
 - A: una sola letra mayúscula
 - mysamplename: minúsculas
 - my_sample_name: minúsculas con guiones bajos (snake_case)
 - MYSAMPLENAME: mayúsculas
 - MY_SAMPLE_NAME: mayúsculas con guiones bajos (SNAKE_CASE)
 - En general, debe evitar usar nombres de una sola letra como l (la letra minúscula el), I (la letra mayúscula eye) y O (la letra mayúscula oh), porque pueden confundirse fácilmente con los números 1 y 0, y hacer que su código sea mucho menos legible.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Estilos (convenciones) de nombres**

- **MySampleName** – CamelCase (también conocido como palabras en mayúsculas, StudlyCaps o CapWords)

Una breve nota: cuando utilice acrónimos, debe escribir en mayúsculas todas las letras que forman el acrónimo, p. ej., HTTPServerError

- **mySampleName** – mayúsculas y minúsculas, que en realidad difiere de CamelCase solo en que tiene un carácter inicial en minúscula
 - **My_Sample_Name** – palabras en mayúsculas con guiones bajos (considerado feo por PEP 8)
 - **_my_sample_name** – un nombre que comienza con un solo guión bajo inicial indica un "uso interno" débil, p. ej., la instrucción **from SAMPLE import *** no importará objetos cuyos nombres comiencen con un guión bajo.
 - **my_sample_name_** – por convención, se utiliza un guión bajo final simple para evitar conflictos con palabras clave de Python, p. ej., `class_`
 - **__my_sample_name** – un nombre que comienza con un guión bajo inicial doble se utiliza para atributos de clase donde invoca la alteración de nombres, p. ej., dentro de la clase `MySampleClass`, `__room` se convertirá en `_MySampleClass__room`
 - **__my_sample_name__** – un nombre que comienza y termina con un guión bajo doble se utiliza para objetos y atributos "mágicos" que residen en espacios de nombres controlados por el usuario, p. ej., `__init__`, `__import__` o `__file__`. No debe crear dichos nombres, sino utilizarlos únicamente como se documenta.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Estilos (convenciones) de nombres. Recomendaciones.**

- PEP 8 proporciona una convención de nombres específica con respecto a un identificador específico (1 de 2).
 - Al dar un nombre a una **variable**, debe utilizar una letra minúscula o una palabra(s) y separar las palabras con guiones bajos, p. ej., x, var, my_variable. La misma convención se aplica a las variables globales.
 - Las **funciones** siguen las mismas reglas que las variables, es decir, al dar un nombre a una función, debe utilizar una letra minúscula o una palabra(s) separadas por guiones bajos, p. ej., fun, my_function.
 - Al dar un nombre a una **clase**, debe adoptar el estilo CamelCase, p. ej., MySampleClass, o si solo hay una palabra, comenzarla con una letra mayúscula, p. ej., Sample.
 - Al dar un nombre a un **método**, debe utilizar una palabra o palabras en minúscula separadas por guiones bajos, p. ej., method, my_class_method. Siempre debe utilizar self como primer argumento para los métodos de instancia y cls como primer argumento para los métodos de clase.
 - Al asignar un nombre a una **constante**, debe utilizar letras mayúsculas y separar las palabras con guiones bajos, por ejemplo, TOTAL, MY_CONSTANT.

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Estilos (convenciones) de nombres. Recomendaciones.**

- PEP 8 proporciona una convención de nombres específica con respecto a un identificador específico (2 de 2).
 - Al dar un nombre a un **módulo**, debe utilizar una o más palabras en minúscula, preferiblemente cortas, y separarlas con guiones bajos, p. ej., `samples.py`, `my_samples..`
 - Al dar un nombre a un **paquete**, debe utilizar una o más palabras en minúscula, preferiblemente cortas. No debe separar las palabras, p. ej., `paquete`, `mipaquete`.
 - Los nombres de las **variables de tipo** deben seguir la convención CamelCase y ser cortos, p. ej., `AnyStr` o `Num`.
 - Al dar un nombre a una **excepción**, debe seguir la misma convención que con las clases (tenga en cuenta que las excepciones deben ser en realidad clases), es decir, utilice el estilo CamelCase.

Nota: Puede utilizar un estilo diferente, p. ej., mayúsculas y minúsculas mixtas (`mySample`) para funciones y variables, pero solo si esto ayuda a mantener la compatibilidad con versiones anteriores y si ese es el estilo predominante.

- Para obtener información más detallada sobre las convenciones de nomenclatura de PEP 8, visite la página oficial de PEP 8:
 - <https://www.python.org/dev/peps/pep-0008/#prescriptive-naming-conventions>

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Programming recommendations.**
 - Realizar comparaciones con el objeto None con el uso de **is** o **is not**, no con los operadores de (des)igualdad (**==** y **!=**).



```
1 # Bad:
2
3 if x == None:
4     print("A")
5
```



```
1 # Good:
2
3 if x is None:
4     print("A")
5
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Programming recommendations.**

- No utilice los operadores de (des)igualdad al comparar valores booleanos con Verdadero o Falso. Nuevamente, utilice `is` o `is not` en su lugar,



```
1 # Bad:
2
3 my_boolean_value = 2 > 1
4 if my_boolean_value == True:
5     print("A")
6 else:
7     print("B")
8
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Programming recommendations.**

- No utilice los operadores de (des)igualdad al comparar valores booleanos con Verdadero o Falso. Nuevamente, utilice `is` o `is not` en su lugar,



```
1 # Good:
2
3 my_boolean_value = 2 > 1
4 if my_boolean_value is True:
5     print("A")
6 else:
7     print("B")
8
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Programming recommendations.**

- No utilice los operadores de (des)igualdad al comparar valores booleanos con Verdadero o Falso. Nuevamente, utilice is o is not en su lugar,



```
1 # Better:
2
3 my_boolean_value = 2 > 1
4 if my_boolean_value:
5     print("A")
6 else:
7     print("B")
8
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Programming recommendations.**

- Utilizar el operador `is not` en lugar de `not...is`.
 - Evitar usar `if x:` para expresar si `x` no es `None`: cuando desee verificar si a una variable o argumento establecido en `None` de manera predeterminada se le ha asignado un valor diferente.



```
1 # Bad:
2
3 if not x is None:
4     print("It exists")
5
```



```
1 # Good:
2
3 if x is not None:
4     print("It exists")
5
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Programming recommendations.**
 - Cuando desee "capturar" una excepción, haga referencia a excepciones específicas en lugar de utilizar únicamente la cláusula `except`.



```
1 try:
2     import my_module
3 except ImportError:
4     my_module = None
5
```

PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**

- **Programming recommendations.**

- Al comprobar prefijos o sufijos, utilice los métodos de cadena `".startswith()"` y `".endswith()"`, ya que son más claros y menos propensos a errores. En general, es mejor utilizar métodos de la clase **string** en lugar de importar el módulo **string**.



```
1 # Bad:
2
3 if name[:4] == 'Adam':
4     # do something
5
```



```
1 # Good:
2
3 if name.startswith('Adam'):
4     # do something
5
```


PEP 8 - STYLE GUIDE FOR PYTHON CODE

- **Recomendaciones para el diseño del código**
 - **Programming recommendations.**
 - Para obtener más sugerencias sobre cómo escribir mejor código y qué prácticas debes evitar, consulta la página oficial de Recomendaciones de programación PEP 8:
 - <https://peps.python.org/pep-0008/#programming-recommendations>