
# Taller PEP8 – Style Guide for Python Code

## Estándares del código en Python

2023-II

Juan Pablo Mejía Gómez

*((*Those who can imagine anything, can create the impossible. (...)*))*

Alan Turing

## Introducción

En este taller aprenderemos las mejores prácticas para escribir código Python elegante, legible y consistente siguiendo las normas de estilo de PEP8. Durante el taller, exploraremos en detalle las directrices recomendadas por la guía de estilos PEP8 para la indentación, el uso de espacios en blanco, los nombres de variables y funciones, los comentarios y docstrings, y cómo evitar código innecesario. A través de ejemplos prácticos y ejercicios de programación, los estudiantes adquirirán las habilidades esenciales para mejorar la calidad y mantenibilidad de su código, lo que los convertirá en programadores más eficientes y colaborativos.

# Contenido

## Indentación y Espacios en Blanco

La correcta indentación y el uso adecuado de espacios en blanco son fundamentales para mejorar la legibilidad del código Python. Esto incluye utilizar 4 espacios por nivel de indentación y evitar el uso de tabulaciones.

## Identificadores

La elección de nombres descriptivos y coherentes para las variables, funciones y clases es esencial para que el código sea comprensible y mantenga la consistencia. Se deben seguir los estilos snake_case para variables y funciones, y CamelCase para clases. Se recomienda declarar estas variables en el idioma inglés.

## Comentarios y Docstrings

Los comentarios y docstrings adecuados proporcionan una documentación valiosa del código. Los comentarios explican el propósito y la lógica detrás del código, mientras que los docstrings proporcionan información sobre funciones, clases y módulos.

## Evitar Código Innecesario

Mantener el código limpio y evitar líneas innecesarias mejora la claridad y facilidad de mantenimiento del código. Al eliminar código redundante o no utilizado, se reduce la complejidad y el riesgo de errores.

## Consistencia en el Estilo

Seguir un estilo de codificación consistente en todo el proyecto es crucial para fomentar la colaboración entre desarrolladores y mejorar la legibilidad del código. La consistencia en el estilo facilita la comprensión del código por parte de todos los miembros del equipo.

Estos cinco elementos abordan los aspectos más importantes para lograr un código limpio, legible y bien estructurado en Python, siguiendo las recomendaciones de PEP 8. Al aplicar estos principios, el

código será más fácil de mantener y comprender, lo que contribuirá a la eficiencia y calidad del desarrollo de proyectos en Python.

# Desarrollo

## Indentación y espacios en blanco

En Python, la indentación y los espacios en blanco son elementos fundamentales de la estructura del código que influyen en la legibilidad y organización del mismo. Según PEP 8, la indentación consiste en la forma en que se colocan los espacios o tabulaciones al principio de una línea de código para denotar la jerarquía y las estructuras de control, como bloques de funciones, bucles y condicionales. En Python, se recomienda utilizar cuatro espacios por nivel de indentación y evitar el uso de tabulaciones para asegurar que el código sea consistente y fácilmente comprensible por otros desarrolladores. La indentación adecuada es esencial en Python, ya que determina la ejecución del código y un error de indentación puede llevar a resultados inesperados.

Los espacios en blanco, también regidos por PEP 8, se refieren al uso de espacios adicionales en diversas partes del código para mejorar la legibilidad y claridad. De acuerdo con PEP 8, se deben agregar espacios en blanco alrededor de operadores, después de comas y dos puntos, y después de palabras clave (if, for, while, etc.). Además, se recomienda limitar la longitud de línea a 79 caracteres y, en caso de líneas largas, dividirlas adecuadamente utilizando paréntesis o el carácter de continuación (\). Estas convenciones mejoran la estructura visual del código, facilitan la identificación de bloques lógicos y hacen que el código sea más fácil de leer y mantener para todos los desarrolladores involucrados en el proyecto.

```python
# Correct: Indentation with four spaces
def print_numbers():
    for i in range(1, 6):
        print(i)

# Incorrect: Indentation with two spaces (not recommended by PEP 8)
def print_letters():
  for letter in 'Python':
    print(letter)
```

En este ejemplo, la función print_numbers tiene una indentación adecuada con cuatro espacios en cada nivel, lo cual sigue las recomendaciones de PEP 8. Sin embargo, la función print_letters tiene una indentación incorrecta con solo dos espacios en cada nivel, lo cual no es recomendado por PEP8.

## Identificadores

En Python, se recomienda utilizar nombres descriptivos y significativos para las variables y funciones, lo que ayuda a comunicar claramente su propósito y funcionalidad. Para los nombres de variables y funciones, se debe seguir el estilo snake_case, que consiste en escribir todas las letras en minúscula y separar las palabras con guiones bajos. Esta convención facilita la lectura y comprensión del código al resaltar las palabras clave y hacer que los nombres sean más fáciles de distinguir.

PEP8 también establece un estilo específico para los nombres de clases, conocido como CamelCase, que consiste en comenzar cada palabra con una letra mayúscula sin espacios entre ellas. Esta convención distingue claramente los nombres de las clases de las variables y funciones, lo que ayuda a mantener una organización coherente en el código. Al adherirse a estas convenciones de nomenclatura, los programadores pueden colaborar más eficientemente en proyectos y facilitar la comprensión de su código tanto para ellos mismos como para otros miembros del equipo. En resumen, el uso de nombres significativos y la adhesión al estilo snake_case y CamelCase, según corresponda, son

prácticas esenciales para escribir código legible y estructurado en Python, tal como lo recomienda PEP 8.

```python
# Correcto: Nombres descriptivos en minúsculas con guiones bajos (snake_case
cantidad_productos = 100
nombre_usuario = "John"

def calcular_promedio(lista_numeros):
    total = sum(lista_numeros)
    return total / len(lista_numeros)
```

En este ejemplo, los nombres de variables y funciones utilizan el estilo snake_case, que consiste en escribir todas las letras en minúscula y separar las palabras con guiones bajos. Esto hace que los identificadores sean más legibles y descriptivos.

```python
# Correct: Descriptive variable names in lowercase with underscores (snake_c
num_items = 100
user_name = "John"

def calculate_average(number_list):
    total = sum(number_list)
    return total / len(number_list)
```

En este caso, el nombre de la clase utiliza el estilo CamelCase, que comienza cada palabra con una letra mayúscula sin espacios entre ellas. Esto ayuda a distinguir claramente las clases de las funciones y variables, siguiendo las convenciones recomendadas por PEP 8.

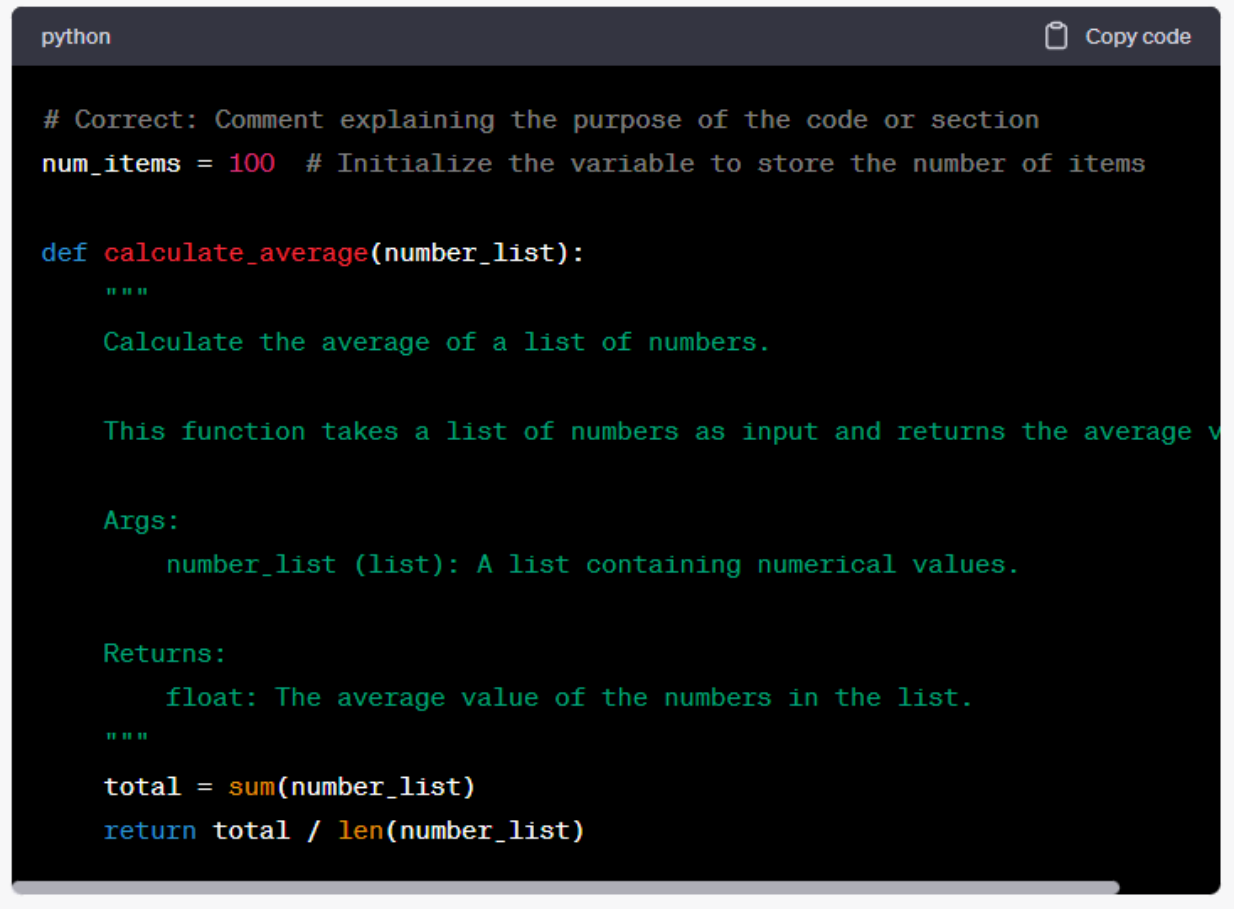
## Comentarios y docstrings

En Python, los comentarios y docstrings son herramientas fundamentales para proporcionar una documentación efectiva del

código, tal como lo indica PEP 8, la guía de estilo para Python. Los comentarios son líneas de texto que se utilizan para explicar fragmentos de código y agregar notas aclaratorias sobre su funcionamiento. Estos comentarios son precedidos por el símbolo # y no afectan la ejecución del código. Su propósito es mejorar la comprensión del código y permitir a otros desarrolladores entender rápidamente la lógica detrás de cada línea.

Por otro lado, los docstrings son cadenas de texto colocadas al principio de una función, clase o módulo. Sirven para proporcionar una descripción más detallada de su propósito, los parámetros que acepta y los valores que devuelve, en el caso de funciones. Los docstrings son accesibles mediante el atributo __doc__ y se utilizan para generar documentación automáticamente y facilitar el trabajo colaborativo en proyectos. Al seguir las convenciones de comentarios y docstrings definidas en PEP 8, los programadores pueden asegurar una comunicación clara y consistente sobre el código, lo que contribuye a la comprensión, el mantenimiento y la reutilización efectiva de sus

proyectos.

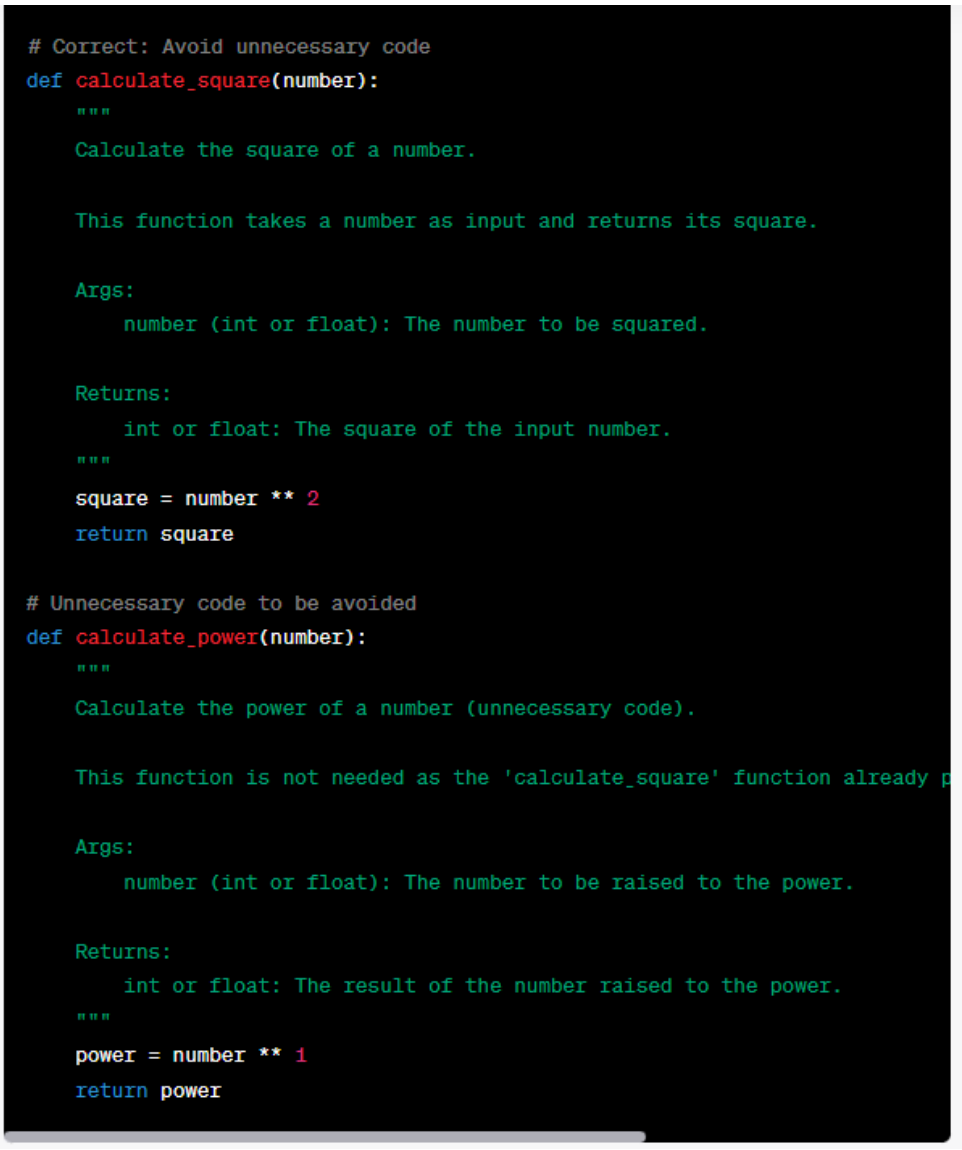



En este ejemplo, el comentario proporciona una explicación breve del propósito del código, y el comentario después de num_items = 100 explica el propósito de la línea específica. La función calculate_average tiene un docstring encerrado entre comillas triples, el cual brinda una descripción detallada de su propósito, los argumentos que acepta y el valor que devuelve. Seguir las pautas de PEP 8 para comentarios y docstrings hace que el código sea más informativo y fácil de mantener.

## Evitar código innecesario

Evitar código innecesario en Python, siguiendo las directrices de PEP 8, implica la práctica de escribir un código conciso y eficiente, evitando líneas o bloques de código que no contribuyen directamente a la funcionalidad o lógica del programa. Eliminar código redundante o superfluo es crucial para mantener un código limpio y fácil de mantener, ya que reduce la complejidad y el riesgo de errores.

PEP 8 enfatiza la importancia de no repetir tareas innecesariamente y de evitar el uso de estructuras o funciones obsoletas o poco eficientes. Al evitar código innecesario, los programadores pueden mejorar el rendimiento del programa, hacerlo más legible y reducir la cantidad de trabajo necesario para realizar futuras actualizaciones o correcciones. Al fomentar la simplicidad y la eficiencia en el código, PEP 8 promueve una mejor práctica de programación y un mayor enfoque en la funcionalidad y mantenibilidad del código Python.



```python
# Correct: Avoid unnecessary code
def calculate_square(number):
    """
    Calculate the square of a number.

    This function takes a number as input and returns its square.

    Args:
        number (int or float): The number to be squared.

    Returns:
        int or float: The square of the input number.
    """
    square = number ** 2
    return square

# Unnecessary code to be avoided
def calculate_power(number):
    """
    Calculate the power of a number (unnecessary code).

    This function is not needed as the 'calculate_square' function already p

    Args:
        number (int or float): The number to be raised to the power.

    Returns:
        int or float: The result of the number raised to the power.
    """
    power = number ** 1
    return power
```

En este ejemplo, la función calculate_power es innecesaria porque realiza el mismo cálculo que la función calculate_square, es decir, elevar

un número a la potencia de 1. Dado que cualquier número elevado a la potencia de 1 es el mismo número, la función calculate_power es redundante y se puede evitar. Al confiar únicamente en la función calculate_square, mantenemos un código más limpio y eficiente, siguiendo los principios de PEP 8.

## Consistencia en el estilo

La consistencia en el estilo, según PEP 8, es un principio fundamental para escribir código en Python que busca mantener una apariencia y estructura uniformes en todo el proyecto. Este enfoque consistente facilita la colaboración entre desarrolladores y mejora la legibilidad del código. Al seguir las mismas convenciones de nombres, indentación, espacios en blanco, comentarios y otros elementos de estilo en todo el código, se crea una experiencia coherente y más fácil de seguir para quienes revisan y mantienen el proyecto.

PEP 8 enfatiza la importancia de que todos los miembros del equipo adhieran al mismo estilo de codificación, incluso si tienen diferentes antecedentes y preferencias personales. Al mantener una base de código coherente, se reduce la confusión y la posibilidad de errores causados por diferencias de estilo. La consistencia también hace que el código sea más predecible y estandarizado, lo que resulta en un desarrollo más eficiente y una mayor facilidad para identificar posibles problemas. En resumen, al seguir las convenciones de estilo establecidas por PEP 8 y mantener la consistencia en todo el proyecto, los programadores pueden crear un código más claro, legible y fácil de mantener para todo el equipo.

```python
# Correct: Consistency in style
def calculate_area_of_rectangle(length, width):
    """
    Calculate the area of a rectangle.

    This function takes the length and width of a rectangle as input and ret

    Args:
        length (float): The length of the rectangle.
        width (float): The width of the rectangle.

    Returns:
        float: The area of the rectangle.
    """
    area = length * width
    return area

# Consistency in style
base = 5.0
height = 10.0
triangle_area = 0.5 * base * height

print(calculate_area_of_rectangle(4.5, 3.2))
print(triangle_area)
```

En este ejemplo, se demuestra la consistencia en el estilo siguiendo las convenciones de PEP 8:

Los nombres de las funciones y variables están escritos en minúsculas con guiones bajos (snake_case). Se utilizan nombres descriptivos para las variables y funciones para hacer que el código sea más legible. Se proporcionan comentarios utilizando el símbolo # para explicar el propósito del código y los parámetros de las funciones. La indentación utiliza cuatro espacios para cada nivel de anidamiento. Al seguir estas convenciones de estilo consistentemente en todo el código, se crea una

estructura más limpia y organizada, lo que facilita la lectura y el mantenimiento del código.

## Ejercicios prácticos

### Ejercicio de Indentación:

Escribe una función que tome una lista de números como entrada y devuelva la suma de todos los números pares en la lista. Asegúrate de utilizar una indentación adecuada para el bloque del bucle y el bloque de la función.

### Ejercicio de Nombres de Variables:

Escribe una función que tome una cadena de texto como entrada y devuelva una versión modificada de la cadena, donde todas las palabras estén en minúsculas y separadas por guiones bajos. Utiliza nombres de variables descriptivos y coherentes.

### Ejercicio de Comentarios y Docstrings:

Escribe una función que tome una lista de nombres como entrada y devuelva una lista nueva con los nombres en orden alfabético. Asegúrate de proporcionar comentarios y docstrings que expliquen el propósito de la función y los detalles sobre los parámetros y el valor de retorno.

### Ejercicio de Espacios en Blanco:

Escribe una función que tome dos números como entrada y devuelva la diferencia entre ellos. Asegúrate de agregar espacios en blanco adecuados alrededor de los operadores y después de los dos puntos en el docstring.

### Ejercicio de Longitud de Línea:

Escribe una función que tome una lista de números como entrada y devuelva el producto de todos los elementos. Asegúrate de mantener la longitud de línea dentro del límite recomendado de 79 caracteres.

### Ejercicio de Evitar Código Innecesario:

Escribe una función que tome una lista de números como entrada y devuelva una lista nueva con solo los números positivos. Evita repetir tareas innecesariamente y optimiza el código para que sea lo más eficiente posible.

### Ejercicio de Consistencia en el Estilo:

Escribe una función que tome una cadena de texto como entrada y devuelva la cadena con todas las vocales en mayúsculas y todas las consonantes en minúsculas. Asegúrate de seguir las mismas convenciones de estilo en toda la función.

### Ejercicio de Nombres de Funciones:

Escribe una función que tome una lista de palabras como entrada y devuelva la palabra más larga en la lista. Utiliza un nombre de función descriptivo que indique claramente su propósito.

### Ejercicio de Organización del Código:

Escribe una función que tome una lista de números como entrada y devuelva una lista nueva con solo los números impares. Organiza el código de manera coherente y utiliza líneas en blanco para separar bloques lógicos.

## Preguntas teóricas

1. ¿A qué corresponden las siglas PEP?
2. ¿Cuál es el propósito de las guías propuestas por Guido van Rossum?
3. ¿Las guías de estilo son siempre aplicables en la escritura de un código? Justifique su respuesta.
4. ¿Cuántos espacios de indentado propone la guía de estilo PEP8?
5. Explique que es un *hanging indent*.

6. Seleccione de entre las dos opciones cuál es la que cumple con la guía de estilos PEP8:

*a)*
```
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

*b)*
```
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

7. ¿Existe una regla específica para el manejo de sentencias condicionales según la guía PEP8? Justifique su respuesta.

8. Seleccione la opción que **NO** cumple con la guía de estilos PEP8 de entre las siguientes opciones:

*a)*
```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
```

*b)*
```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
```

*c)*
```
my_list = [1, 2, 3,
           4, 5, 6,]
```

9. ¿Cuál es el método de indentación preferido según PEP8?

10. ¿En qué casos se utilizan *Tabs* para indentar código según PEP8?

11. ¿Pueden mezclarse *Tabs* y *Spaces* para indentar según PEP8?

12. ¿Cuál es el máximo número de caracteres para código según PEP8? ¿Cuál es el número máximo para comentarios y/o documentación? ¿Es

posible aumentar el número de caracteres? De ser así, mencione en qué casos.

13. Enuncie alguna ventaja de limitar el número de caracteres.

14. ¿Por qué PEP8 no recomienda el uso de *wrapping* en editores de código?

15. ¿Cuál es el método preferido para realizar *wrapping* de largas líneas según PEP8?

16. Seleccione la opción que cumple con la guía de estilos PEP8 de entre las siguientes opciones:

*a)*

```
income = (gross_wages +
          taxable_interest +
          (dividends - qualified_dividends) -
          ira_deduction -
          student_loan_interest)
```

*b)*

```
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

17. ¿Se permite romper una línea antes y/o después de un operador binario en Python según PEP8? ¿Cuál es el método más sugerido?

18. ¿En qué casos rodeamos la declaración de clases o de funciones con dos (2) líneas en blanco según PEP8? ¿En qué casos lo hacemos con una (1) línea en blanco según PEP8?

19. ¿Cuándo podemos omitir las líneas en blanco dentro de Python según PEP8

20. ¿Cuál es el estándar de codificación de caracteres más recomendado por PEP8 para Python?

21. ¿En qué idioma natural preferiblemente debemos declarar identificadores (variables, clases, funciones, entre otros) en el código

Python según PEP8? ¿Cuál es el estándar de representación de caracteres que deben usar estos obligatoriamente?

22. Seleccione la opción que **NO** cumple con la guía de estilos PEP8 de entre las siguientes opciones:

    *a)*
    ```
    import os
    import sys
    ```

    *b)*
    ```
    import sys, os
    ```

    *c)*
    ```
    from subprocess import Popen, PIPE
    ```

23. Ordene los siguientes grupos de importación según el orden de importación de PEP8: Standard library imports, Related third party imports, Local application/library specific imports

24. ¿Cuál es el método de importación más recomendado según PEP8? Enuncie en qué casos se utiliza la importación absoluta y la importación relativa.

25. ¿Se recomienda utilizar el operador asterisco (*) para realizar importaciones según PEP8?

26. ¿Dónde se deben poner las declaraciones de identificadores *dunders* dentro del código Python según PEP8? ¿Cuáles son las excepciones?

27. ¿Pueden usarse comillas simples o comillas dobles, sin limitaciones, para escribir cadenas de texto en Python según PEP8?