PYTHON

Best Practice and Standarization

PEP 257, PEP 484, Documentation standards and linters

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **¿Qué es PEP 257?**
- PEP 257 es un documento creado como parte de la Guía del desarrollador de Python, que intenta estandarizar la estructura de alto nivel de las cadenas de documentación. Describe las convenciones, las mejores prácticas y la semántica asociadas con la documentación del código Python mediante cadenas de documentación. En resumen, intenta responder las dos preguntas siguientes:
  - ¿Qué deben contener las cadenas de documentación de Python?
  - ¿Cómo se deben utilizar las cadenas de documentación de Python?

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **¿Qué son las cadenas de documentación?**
  - Una cadena de documentación es "una cadena literal que aparece como la primera declaración en una definición de módulo, función, clase o método. Dicha cadena de documentación se convierte en el atributo especial __doc__ de ese objeto". (PEP 257)
  - En otras palabras, las cadenas de documentación son cadenas de documentación de Python que se utilizan en la definición de clase, módulo, función y método para proporcionar información sobre la funcionalidad de un fragmento de código más grande de forma prescriptiva.
  - Ayudan a los programadores a recordar y comprender el propósito, el funcionamiento y las capacidades de bloques o secciones de código particulares.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Docstrings vs. comentarios**
- Antes de continuar, debemos entender esta distinción esencial (como sugieren sus nombres): los comentarios se utilizan para comentar el código, mientras que las docstrings se utilizan para documentarlo. Entonces, ¿cuál es la diferencia entre comentarios y docstrings y, en última instancia, entre comentar y documentar código?

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Comentarios**
  - Los comentarios son declaraciones no ejecutables en Python, lo que significa que el intérprete de Python los ignora; no se almacenan en la memoria y no se puede acceder a ellos durante la ejecución del programa (es decir, se puede acceder a ellos mirando el código fuente).
  - El objetivo principal de los comentarios es aumentar la legibilidad y la comprensión del código y explicar el código al usuario de una manera significativa. El usuario aquí se refiere tanto a otros programadores como a usted (por ejemplo, cuando vuelve a su código después de un tiempo), alguien que querrá o necesitará modificar, ampliar o mantener el código.
  - Los comentarios no se pueden convertir en documentación; su propósito es simplificar el código, proporcionar información precisa y ayudar a comprender la intención de un fragmento o una línea en particular.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cadenas de documentación (docstrings)**
- Se puede acceder a las cadenas de documentación leyendo el código fuente y utilizando el atributo __doc__ o la función help().
- El objetivo principal de las cadenas de documentación es documentar el código, es decir, describir su uso, funcionalidad y capacidades a usuarios que no necesariamente necesitan saber cómo funciona.
- Las cadenas de documentación se pueden convertir fácilmente en documentación real, que describe el comportamiento de un módulo o una función, el significado de los parámetros o el propósito de un paquete específico.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **¿Por qué comentar? ¿Por qué documentar?**
- Antes de profundizar en el tema de las cadenas de documentación, intentemos responder a la pregunta: ¿por qué es importante comentar y documentar el código?
- Básicamente, no debemos olvidar esta sencilla regla de Guido van Rossum: "El código se lee más a menudo de lo que se escribe", lo que básicamente significa que el código que escribimos hoy probablemente será leído en el futuro, ya sea por usted, por otro programador o incluso por equipos de programadores.
- Por lo tanto, es fundamental que desarrollemos hábitos de programación y escritura de código que permitan a los desarrolladores y otros usuarios comprender los porqués y los cómos del código, ya que esto hará que la reutilización y la contribución al código sean mucho más fáciles.
- Por lo tanto, deberíamos estar de acuerdo en que documentar el código ayuda a mantener un código más limpio, más legible y más sostenible, lo que significa que es una de las mejores prácticas que un desarrollador responsable y bueno debería adoptar como parte du conjunto de herramientas de taller de programación diaria.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Un breve resumen de los comentarios**
- Esperamos que recuerdes que los comentarios en Python se crean usando el signo almohadilla (#). Deben ser bastante breves (no más de 72 caracteres por línea), comenzar con una letra mayúscula y terminar con un punto.
- Si necesitas incluir un comentario más largo en tu código, puedes usar un comentario de varias líneas, en cuyo caso debes usar el signo almohadilla al comienzo de cada línea de comentario.
- Generalmente, debes insertar comentarios cerca del código que estás describiendo para que el lector sepa claramente a qué parte del código te estás refiriendo. Debes ser preciso: no incluyas información irrelevante o redundante; y, sobre todo, intenta diseñar y escribir tu código de tal manera que se comente a sí mismo de manera fácil y comprensible (por ejemplo, dale nombres de autocomentarios a las variables).

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Un breve resumen de los comentarios**
- Esperamos que recuerdes que los comentarios en Python se crean usando el signo almohadilla (#). Deben ser bastante breves (no más de 72 caracteres por línea), comenzar con una letra mayúscula y terminar con un punto.
- Si necesitas incluir un comentario más largo en tu código, puedes usar un comentario de varias líneas, en cuyo caso debes usar el signo almohadilla al comienzo de cada línea de comentario.
- Generalmente, debes insertar comentarios cerca del código que estás describiendo para que el lector sepa claramente a qué parte del código te estás refiriendo. Debes ser preciso: no incluyas información irrelevante o redundante; y, sobre todo, intenta diseñar y escribir tu código de tal manera que se comente a sí mismo de manera fácil y comprensible (por ejemplo, dale nombres de autocomentarios a las variables).

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **¿Cuándo se utilizan los comentarios?**
- Además de los casos más obvios, como las descripciones de código y algoritmos, los comentarios pueden tener otros propósitos útiles. Por ejemplo:
- (l) Pueden ayudarle a etiquetar aquellas secciones de código que se realizarán en el futuro o que se dejaron para mejoras posteriores, por ejemplo:

```
# TODO: Add a function that takes the val and prc arguments.
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **¿Cuándo se utilizan los comentarios?**
- (2) Pueden ayudarte a comentar (y descomentar) aquellas secciones de código que quieras probar, por ejemplo:

```
def fun(val):
return val * 2
user_value = int(input("Enter the value: "))
# fun(user_value)
# user_value = user_value + "foo"
print(fun(user_value))
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **¿Cuándo se utilizan los comentarios?**
- (3) Pueden ayudarte a planificar tu trabajo y delinear ciertas secciones del código que estarás diseñando, por ejemplo:

```
# Step 1: Ask the user for the value.
# Step 2: Change the value to an int and handle possible exceptions.
# Step 3: Print the value multiplied by 0.7.
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Algunas palabras sobre sugerencias de tipos (type hints): PEP 484**
- Las sugerencias de tipo son un mecanismo introducido con Python 3.5 y descrito en PEP 484 que te permite equipar tu código con información adicional sin usar comentarios. Es una característica opcional, pero más formalizada, que te permite usar el módulo de tipado integrado de Python para proporcionar información de sugerencias de tipo en tu código con el fin de dejar ciertas sugerencias, marcar ciertos posibles problemas que puedan surgir en el proceso de desarrollo y etiquetar nombres específicos con información de tipo.
- En pocas palabras, las sugerencias de tipo te permiten indicar estáticamente la información de tipo relacionada con los objetos de Python, lo que significa que puedes, por ejemplo, agregar información de tipo a una función: indicar el tipo de un argumento que acepta la función o el tipo de valor que devolverá.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Algunas palabras sobre sugerencias de tipos (type hints): PEP 484**

```
# No type information added:
def hello(name):
return "Hello, " + name
# Type information added to a function:
def hello(name: str) -> str:
return "Hello, " + name
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Algunas palabras sobre sugerencias de tipos (type hints): PEP 484**
- La sugerencia de tipo es opcional, lo que significa que PEP 484 no lo obliga a dejar ninguna información relacionada con el tipado estático en su código. El primer ejemplo no tiene ninguna sugerencia de tipo.
- En el segundo ejemplo, la anotación str, que indica que el argumento de nombre pasado a la función hello() debe ser del tipo str, nos ayuda a minimizar el riesgo de ciertas situaciones (in)esperadas: reduce el riesgo de pasar un tipo de valor incorrecto a la función. La anotación -> str también indica que la función hello() devolverá un valor de tipo str, que, por supuesto, es una cadena.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Algunas palabras sobre sugerencias de tipos (type hints): PEP 484**
- La sugerencia de tipo es opcional, lo que significa que PEP 484 no lo obliga a dejar ninguna información relacionada con el tipado estático en su código. El primer ejemplo no tiene ninguna sugerencia de tipo.
- En el segundo ejemplo, la anotación str, que indica que el argumento de nombre pasado a la función hello() debe ser del tipo str, nos ayuda a minimizar el riesgo de ciertas situaciones (in)esperadas: reduce el riesgo de pasar un tipo de valor incorrecto a la función. La anotación -> str también indica que la función hello() devolverá un valor de tipo str, que, por supuesto, es una cadena.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- Las sugerencias de tipos pueden ayudar a documentar su código. En lugar de dejar información relacionada con argumentos y respuestas en cadenas de documentación, puede usar el lenguaje en sí para cumplir con este propósito. Esta puede ser una forma elegante y útil de resaltar parte de la información de código más importante, especialmente al publicar código en un proyecto, compartirlo con otros desarrolladores o dejar sugerencias para usted mismo cuando tenga que volver al código fuente en el futuro. En algunos de los proyectos de desarrollo de software más grandes, las sugerencias de tipos son una práctica recomendada que ayuda a los equipos a comprender mejor las formas en que los tipos se ejecutan a través del código.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- Las sugerencias de tipos le permiten notar ciertos tipos de errores de manera más efectiva y escribir un código más atractivo y, sobre todo, más limpio. Al usar sugerencias de tipos, piensa con más cuidado en los tipos en su código, lo que ayuda a prevenir o detectar algunos de los errores que pueden resultar de la naturaleza dinámica de Python.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- Las sugerencias de tipo en Python no se utilizan en tiempo de ejecución, lo que significa que toda la información de tipo que dejas en el código en forma de anotaciones se borra cuando se ejecuta el programa. En otras palabras, las sugerencias de tipo no tienen ningún efecto en el funcionamiento de tu código.
- Cuando se utilizan junto con algún sistema de comprobación de tipo o herramientas similares a lint que puedes conectar a tu editor o IDE, pueden ayudar a escribir tu código completando automáticamente tu tipeo y detectando y resaltando errores antes de que se ejecute tu código.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- Las sugerencias de tipo en Python no se utilizan en tiempo de ejecución, lo que significa que toda la información de tipo que dejas en el código en forma de anotaciones se borra cuando se ejecuta el programa. En otras palabras, las sugerencias de tipo no tienen ningún efecto en el funcionamiento de tu código.
- Cuando se utilizan junto con algún sistema de comprobación de tipo o herramientas similares a lint que puedes conectar a tu editor o IDE, pueden ayudar a escribir tu código completando automáticamente tu tipeo y detectando y resaltando errores antes de que se ejecute tu código.
- No tienen influencia en los tiempos de ejecución.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- Para obtener más información sobre el tema, le recomendamos que eche un vistazo más de cerca a **PEP 483**: la teoría de las sugerencias de tipos, **PEP 484**: sugerencias de tipos (información sobre la sintaxis para anotaciones de tipos, análisis estático y refactorización, verificación de tipos) y **PEP 3107**: anotaciones de funciones (información sobre la sintaxis para agregar anotaciones de metadatos a las funciones de Python).

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Doc string: ¿dónde y cómo?**
- Las cadenas de documentación se pueden usar en clases, módulos, funciones y definiciones de métodos. Ahora queremos profundizar en esto: hay casos en los que no solo se pueden incluir, sino que se deben incluir. Para ser más precisos, todos los módulos, funciones, clases y métodos públicos que se exportan por un módulo determinado deben tener cadenas de documentación.
- Los métodos no públicos no necesitan contener cadenas de documentación. Sin embargo, se recomienda que deje un comentario justo después de la línea def que describa lo que realmente hace el método. En el caso de los paquetes, estos también deben documentarse y puede escribir cadenas de documentación de paquetes en la cadena de documentación del módulo del archivo __init__.py en la carpeta del paquete.
- Las cadenas de documentación son literales de cadena que aparecen como la primera declaración en un módulo, función, clase o método. Sin embargo, es importante (y justo) agregar que las cadenas literales de cadena también pueden aparecer en muchos otros lugares en el código Python y aún así servir como documentación. Y aunque ya no sean accesibles como atributos de objetos de tiempo de ejecución, aún pueden extraerse mediante algunas herramientas de software específicas (para obtener más información sobre éstas, consulte PEP 256, que proporciona información sobre ***Docutils***, un sistema de procesamiento de documentos de Python).

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Dos tipos de "docstrings adicionales:**
- **docstrings de atributos**, que se encuentran inmediatamente después de una declaración de asignación en el nivel superior de un módulo (atributos de módulo), clase (atributos de clase) o la definición del método __init__ de una clase (atributos de instancia). Estas son interpretadas por las herramientas de extracción, como ***Docutils***, como "docstrings del destino de la declaración de asignación". (Si estás interesado en aprender más sobre los docstrings de atributos, puedes hacerlo en PEP 224).
- **docstrings adicionales**, que se encuentran inmediatamente después de otro docstring. (La idea original para los docstrings adicionales fue tomada de PEP 216, que a su vez fue reemplazado posteriormente por PEP 287).

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cómo crear docstrings:**
  - Las cadenas de documentación deben estar entre comillas dobles triples ("""comillas dobles triples""").

```
def my_function():
"""I am a docstring."""
...
```

  - Si necesita utilizar barras invertidas en sus cadenas de documentación, debe seguir el formato **r"""comillas dobles triples sin formato"""**. Si necesita utilizar cadenas de documentación Unicode, siga el formato **u"""Cadenas de comillas triples Unicode"""**.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cadenas de documentación de una línea frente a cadenas de documentación de varias líneas:**
  - Cadenas de documentación de una línea: se utilizan para descripciones simples y breves y deben caber en una línea;

Single-line docstring:

```
def my_function():
    """One-line description."""
    body_of_the_function
    ...
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cadenas de documentación de una línea frente a cadenas de documentación de varias líneas:**
  - Cadenas de documentación de varias líneas: se utilizan para casos más difíciles y deben constar de una línea de resumen seguida de una línea en blanco y una descripción más elaborada.;

Multi-line docstring:

```
def my_function(a,b,c):
    """Summary line followed by a blank line

    More elaborate description.
    ...
    ...
    """
    body_of_the_function
    ...
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cadenas de documentación de una línea**
- Las cadenas de documentación de una línea se deben utilizar para descripciones bastante simples, obvias y breves. Deben ocupar solo una línea y estar rodeadas de comillas dobles triples (las comillas de cierre deben estar en la misma línea que las comillas de apertura, ya que esto ayuda a mantener el código limpio y elegante).

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

Cadenas de documentación de una línea

- Notas importantes:
  - Una cadena de documentación debe comenzar con una letra mayúscula (a menos que un identificador comience la oración) y terminar con un punto;
  - Una cadena de documentación debe prescribir el efecto del segmento de código, no describirlo. En otras palabras, debe adoptar la forma de un imperativo.

```
def greeting(name):
"""Take a name and return its replicated form."""
return name * 2
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Una cadena de documentación no debe simplemente repetir los parámetros de la función o del método.**

✖ Don't do this:

```
def my_function(x, y):
    """my_function(x, y) -> list"""
...
```

✔ Instead, try to do something like this:

```
def my_function(x, y):
    """Compute the angles and return a list of coordinates."""
...
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **No utilice una línea en blanco encima o debajo de una cadena de documentación de una línea a menos que esté documentando una clase, en cuyo caso debe colocar una línea en blanco después de todas las cadenas de documentación que la documentan:**

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Docstring multilínea:**
- Las cadenas de documentación de varias líneas se deben utilizar para casos no obvios y descripciones más detalladas de segmentos de código. Deben tener una línea de resumen, similar a la que tiene una cadena de documentación de una sola línea, seguida de una línea en blanco y una descripción más elaborada. La línea de resumen puede estar ubicada en la misma línea que las comillas dobles triples de apertura o en la línea siguiente. Las comillas finales se deben colocar en una línea separada.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Docstring multilínea:**
- Notas importantes:
  - Una cadena de documentación de varias líneas debe tener la misma sangría que las comillas de apertura, por ejemplo:

```
def king_creator(name="Greg", ordinal="I", country="Neverland"):
    """Create a king following the article title naming convention.

    Keyword arguments:
    :arg name: the king's name (default: Greg)
    :type name: str
    :arg ordinal: Roman ordinal number (default: I)
    :type ordinal: str
    :arg country: the country ruled (default: Neverland)
    :type country: str
    """
    if name == "Voldemort":
        return "Voldemort is a reserved name."
    ...
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Docstring multilínea:**
- Notas importantes:
  - Debes insertar una línea en blanco después de todas las cadenas de documentación de varias líneas que documentan una clase;
  - Las cadenas de documentación de los scripts (en el sentido de programas independientes o archivos ejecutables de un solo archivo) deben documentar la función del script, la sintaxis de la línea de comandos, las variables de entorno y los archivos. La descripción debe ser equilibrada de forma que ayude a los nuevos usuarios a comprender el uso del script, así como proporcionar una referencia rápida a todas las características del programa para los usuarios más experimentados;
  - Las cadenas de documentación de los módulos deben incluir las clases, excepciones y funciones exportadas por el módulo;
  - Las cadenas de documentación de los paquetes (entendidas como la cadena de documentación del módulo __init__.py del paquete) deben incluir los módulos y subpaquetes exportados por el paquete;
  - Las cadenas de documentación de las funciones y los métodos de clase deben resumir su comportamiento y proporcionar información sobre los argumentos (incluidos los argumentos opcionales), los valores, las excepciones, las restricciones, etc.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Docstring multilínea:**
- Notas importantes:
  - Las cadenas de documentación de la clase también deben resumir su comportamiento, así como documentar los métodos públicos y las variables de instancia. Por ejemplo:

```
class Vehicle:
    """A class to represent a Vehicle.

    Attributes:
    -----------
    vehicle_type: str
        The type of the vehicle, e.g. a car.
    id_number: int
        The vehicle identification number.
    is_autonomous: bool
        self-driving -> True, not self-driving -> False
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Docstring multilínea:**

```
Methods:
--------
report_location(lon=45.00, lat=90.00)
    Print the vehicle id number and its current location.
    (default longitude=45.00, default latitude=90.00)
"""
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Docstring multilínea:**

```
def __init__(self, vehicle_type, id_number, is_autonomous=True):
    """
    Parameters:
    -----------
    vehicle_type: str
        The type of the vehicle, e.g. a car.
    id_number: int
        The vehicle identification number.
    is_autonomous: bool, optional
        self-driving -> True (default), not self-driving -> False
    """

    self.vehicle_type = vehicle_type
    self.id_number = id_number
    self.is_autonomous = is_autonomous
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Docstring multilínea:**

```
def report_location(self, id_number, lon=45.00, lat=90.00):
    """
    Print the vehicle id number and its current location.

    Parameters:
    -----------
    id_number: int
        The vehicle identification number.
    lon: float, optional
        The vehicle's current longitude (default is 45.00)
    lat: float, optional
        The vehicle's current latitude (default is 90.00)
    """

...
...
...
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

Cómo documentar un proyecto

- Al documentar un proyecto Python, dependiendo de la naturaleza del proyecto (es decir, privado, compartido, público, de código abierto/dominio público), primero y ante todo debes definir a sus usuarios y pensar en sus necesidades. Crear un perfil de usuario puede resultar útil en este caso, ya que te ayudará a identificar las formas en que los usuarios usarán tu proyecto. Esto significa que puedes mejorar fácilmente su experiencia pensando en cómo van a utilizar tu código e intentando predecir los problemas más comunes que pueden encontrar al hacerlo.
- En general, un proyecto debe contener los siguientes elementos de documentación:
  - un archivo **readme**, que proporciona un breve resumen del proyecto, su propósito y posiblemente algunas pautas de instalación;
  - un archivo **examples.py**, que es un script que muestra algunos ejemplos de cómo utilizar el proyecto;
  - una **licencia** en forma de archivo txt (particularmente importante para proyectos de código abierto y dominio público)
  - **un archivo de cómo contribuir** que proporciona información sobre las posibles formas de contribuir al proyecto (proyectos compartidos, de código abierto y de dominio público).

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cómo documentar un proyecto**
- Como documentar el código puede ser una actividad bastante agotadora y que requiere mucho tiempo, se recomienda encarecidamente que utilice algunas de las herramientas que podrían ayudarlo a generar automáticamente la documentación en el formato deseado y a gestionar las actualizaciones y el control de versiones de la documentación de una manera eficaz y eficiente.
- Hay muchas herramientas y recursos de documentación disponibles, como **Sphinx**, que ya hemos mencionado, o el popular **pdoc**, y muchos más. Le recomendamos que siga este camino.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Linters and fixers**
- Un *linter* es una herramienta que lo ayuda a escribir su código, porque lo analiza para detectar anomalías de estilo y errores de programación en comparación con un conjunto de reglas predefinidas. En otras palabras, es un programa que analiza tu código e informa problemas como errores estructurales y de sintaxis, rupturas de consistencia y falta de compatibilidad con las mejores prácticas o pautas de estilo de código como PEP 8. Los linters más populares son: **Flake8**, **Pylint**, **Pyflakes**, **Pychecker**, **Mypy** y **Pycodestyle** (anteriormente Pep8): la herramienta linter **oficial** para verificar el código Python con las convenciones de PEP 8.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Linters and fixers**
- Un *fixer* es un programa que te ayuda a solucionar estos problemas y a formatear tu código para que sea coherente con los estándares adoptados. Los fixers más populares son: **Black**, **YAPF** y **autopep8**.
- La mayoría de los editores e IDE admiten linters, lo que significa que puedes ejecutarlos en segundo plano mientras escribes código. Esto permite detectar, resaltar e identificar muchas áreas problemáticas en tu código, como errores tipográficos, problemas de tabulación y sangría incorrectos, llamadas de función con la cantidad incorrecta de argumentos, inconsistencias estilísticas, patrones de código peligrosos y muchos más, y formatear automáticamente tu código según una especificación predefinida.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cómo acceder a las cadenas de documentación**
  - Mediante el atributo __doc__ de Python: si hay cadenas literales después de la definición de una función/módulo/clase/método, se asocian con el objeto como su atributo __doc__, y este atributo proporciona la documentación de ese objeto.

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cómo acceder a las cadenas de documentación**
  - Dado este código.

```
def my_fun(a, b):
    """The summary line goes here.

    A more elaborate description of the function.

    Parameters:
    a: int (description)
    b: int (description)

    Returns:
    int: Description of the return value.
    """
    return a*b

print(my_fun.__doc__)
```

PEP 257, PEP 484, DOCUMENTATION STANDARDS AND LINTERS

- **Cómo acceder a las cadenas de documentación**
  - El acceso a la documentación se puede hacer:
    - print(my_fun.__doc__)
    - help(my_fun)