3. Diseño del código

3.1. Identación

Usar 4 espacios por nivel de identación. No mezclar espacios y tabuladores.

```
# Usando 4 espacios
def es_par(num):
    if num % 2 == 0:
        return True
    return False
```

Longitud de líneas

La longitud de las líneas es un tema controvertido y permite cierta flexibilidad:

- 79 caracteres por línea por defecto y para la librería estándar.
- 72 caracteres por línea de comentarios o documentación.
- 99 caracteres para equipos que acuerden utilizar líneas más largas pero la limitación de comentarios y documentación sigue siendo 72.

El propósito principal de estas limitaciones es **posibilitar el uso de dos ventanas de un editor**, una al lado de la otra, sin que tenga que añadirse saltos de líneas por el editor, **mejorando así la lectura**.

Posición de los operadores

Cuando se tienen muchos operadores en nuevas líneas, las líneas deben de comenzar con el operador.

Posición de líneas en blanco

Cuando se definen conceptos y lógica en Python, es relevante el uso de líneas en blanco a añadir.

- **2 líneas en blanco**: rodeando a funciones de nivel principal y definición de clases.
- **1 línea en blanco**: definición de métodos dentro de una clase.
- **1 línea extra**: entre bloques lógicos de código diferentes.
- **1 línea en blanco al final del fichero**: todos los módulos en Python deben de terminar con una línea en blanco al final de cada módulo si tienen contenido.
- **control-L (^L)**: se utiliza para separar páginas, aunque algunos editores de texto lo omiten.

Importaciones de código

Para las importaciones de paquetes existen algunas reglas también.

- El uso de import debe de ser en líneas separadas.

- El orden de importación es: librería estándar, librerías de terceros, librerías propias o locales.
- Las importaciones absolutas son los recomendados.
- Las importaciones relativas solo pueden ser explícitas y preferiblemente simples y cortas.
- Importaciones usando * (wildcard) deben de ser evitadas.

Variables de módulo

Las variables de módulo definen ciertos aspectos generales:

- __all__: define qué objetos se importarán al hacer uso de import *.
- __author__: define el autor del módulo.
- __version__: define la versión del módulo actual.

Todas estas definiciones han de hacerse al comienzo del módulo, justo debajo del bloque de importaciones.

Cadenas de caracteres

Python soporta diferentes tipos de caracteres para crear cadenas de caracteres siendo simples o triples, comillas simples o dobles.

- **Comillas simples o dobles**: para cadenas de caracteres de una línea o varias.
- **Triples comillas dobles**: para cadenas de documentación y bloques de cadenas de caracteres.

Para escapar con facilidad las comillas simples o dobles se pueden utilizar un tipo u otro.

Espacios en blanco

Los espacios en blanco en Python se utilizan para aclarar o remarcar partes del código, pero hay que hacer un uso contenido de los mismos definiendo las siguientes reglas:

- **Añadir espacios:** entre elementos separados por comas, en los lados de una evaluación operacional, en una asignación o separando bloques de operaciones.
- **Evitar espacios**: en slicings, entre los nombres de funciones y la llamada, entre las variables y el acceso interno usando [], operaciones lógicamente unidas o en paréntesis vacíos.

```
# Bien
mi_funcion(param1, [1, 2])

a += 1
i = i + 1
b = a*2 - 1
c = (a+i) * (a-i)

if a == b:
```

```
    print(a + 'es igual que' + b)

lst[1:2], lst[4:], lst[5:34:2]
lst[inicio + mid : mid + fin]
lst[inicio::fin]

gato = 1
perro = 2
elefante_marron = 4

# Mal
mi_funcion ( param1, [  1,   2] )

a +=1
i=i+1
b = a * 2 - 1
c = (a + i) * (a - i)

if a==b:
   print (a+'es igual que'          +   b)

lst[ 1:2 ], lst[ 4:], lst[5 : 34 : 2]
lst[inicio + mid:mid + fin]
lst[inicio : : fin]

gato            = 1
perro           = 2
elefante_marron = 4
```

Comas finales

El uso de comas al final de una variable debe de hacerse en una nueva línea, y hay que recordar que **las comas al final de una variable crean tuplas, por tanto hacerlas explícitamente es mejor**.

```
# Bien

variable = ('123',)

mi_funcion (param1,

      param2,

      )

# Mal

variable = '123',  # creará una tupla de forma implícita

mi_funcion (param1, param2,)  # la última coma sobra
```

Otra buena razon por la que en listas y tuplas es bueno dejar la coma al final, si el lenguaje lo soporta, es para que **cuando se editen añadiendo nuevos elementos, no se modifica la línea que ya existia, dejando intacto el historial de cambios**.

Comentarios

La PEP 8 contempla muchas reglas sobre cómo tratar los comentarios que se pueden ver a continuación:

- Los comentarios pueden ser contradictorios, y requieren que se actualicen con el código, por tanto, si se pueden evitar haciendo código más claro, mejor.
- Deben de componerse de frases completas.
- Deben de comenzar en mayúscula a no ser que comiencen con el nombre de un identificador, el cual se respetará siempre su tamaño.
- Los bloques de comentarios se componen de varios párrafos terminados en puntos, y terminar en dos espacios en blancos salvo la última frase.
- Los comentarios deben de ser en Ingles a no ser «que se esté 120%» seguro de que se leerán siempre en otro idioma y serán claros.
- Los bloques se aplican al código justo después de donde se encuentran y con la misma identación.
- Los comentarios de línea comienzan con al menos 2 espacios y un # y un espacio después.
- Las cadenas de documentación (docstrings) deben de añadirse en clases, funciones y métodos que sean públicos siguiendo el PEP 257.

Convenciones de nombres

8.1. Principio rector

Elegir nombres que sean descriptivos y fáciles de entender.

8.2. Estilos de nombres descriptivos

Usar estilos de nombres descriptivos para funciones, variables, clases, etc.

El Libro De Python

8.3. Convenciones de nombres prescriptivos

Seguir las convenciones de nombres prescriptivos para funciones, variables, clases, etc. Academia

8.4. Nombres a evitar

Evitar nombres que puedan ser confusos o ambiguos.

8.5. Compatibilidad con ASCII

Usar solo caracteres ASCII en los nombres.

8.6. Nombres de paquetes y módulos

Usar nombres de paquetes y módulos que sean cortos y en minúsculas.

8.7. Nombres de clases

Usar el estilo CapitalizedWords para los nombres de clases.

8.8. Nombres de variables y funciones

- Usar el estilo lower_case_with_underscores para los nombres de variables y funciones.FreeCodeCamp+5GitHub+5programacionpython.ecyt.unsam.edu.ar+5
- **8.9. Nombres de argumentos de funciones y métodos**
- Usar el estilo lower_case_with_underscores para los nombres de los argumentos de funciones y métodos.
- **8.10. Nombres de métodos y variables de instancia**
- Usar el estilo lower_case_with_underscores para los nombres de métodos y variables de instancia.
- **8.11. Nombres de constantes**
- Usar el estilo UPPER_CASE_WITH_UNDERSCORES para los nombres de constantes.
- **8.12. Diseño para la herencia**
- Diseñar las clases para que sean fáciles de heredar y extender.
- **8.13. Interfaces públicas e internas**
- Distinguir entre interfaces públicas e internas y documentarlas adecuadamente.

Anotaciones de variables

Desde la PEP 526 se introdujeron las anotaciones para las variables y se escriben como:

- **Declaracion**: <nombre_variable>: <tipo>
- **Declaracion + inicializacion**: <nombre_variable>: <tipo> = <valor_por_defecto>

¿Como detectar errores de PEP 8 automáticamente?

Existen librerías que permite detectar errores/violaciones de reglas del PEP 8 denominadas linters y suelen estar incluidas en IDEs de Python (por ejemplo en PyCharm)
Algunos IDEs actuales pueden sugerir el cambio necesario automáticamente antes de realizar un commit en Git.

Se puede utilizar el programa pycodestyle (anteriormente llamado pep8), para comprobar si el código sigue las reglas de correctamente. Este programa se puede instalar usando pip y ejecutarse en consola sobre cualquier módulo python.

```
$ pip install pycodestyle

$ pycodestyle --first nombre_fichero.py

...

$ pycodestyle --show-source --show-pep8 nombre_fichero.py

...

$ pycodestyle --statistics -qq nombre_fichero.py

...
```