pep8_compartir

October 31, 2021

PEP8 PYTHON

Declaraciones y llamada a función

Agregamos una indentación extra de cuatro espacios para distingur entre los argumentos y el contenido de la función.

[1]:
```
def funcion_de_nombre_largo(
        parametro1,
        parametro2,
        parametro3,
        parametro4):
    print(parametro1)
```

Alineamos con la apertura del delimitador en este caso con el paréntesis.

[2]:
```
variable = funcion_de_nombre_largo("Manzana",
                                   "Pera",
                                   "Limón",
                                   "Naranja"
                                   )
```

```
Manzana
```

Se permite también

[3]:
```
variable2 = funcion_de_nombre_largo(
    "Manzana",
    "Pera",
    "Limón",
    "Naranja"
)
```

```
Manzana
```

Cierre de una asignación

El paréntesis / corchete / llave que cierre una asignación debe estar alineado con el primer carácter que no sea un espacio en blanco:

[4]:
```
lista_1 = [
    1, 2, 3,
```

```
    4, 5, 6,
    ]

variable = funcion_de_nombre_largo("Manzana",
                                   "Pera",
                                   "Limón",
                                   "Naranja"
                                   )
```

```
Manzana
```

O puede ser alineado con el caracter inicial de la primera línea

[5]:
```
my_list = [
    1, 2, 3,
    4, 5, 6,
]

variable = funcion_de_nombre_largo(
    "Manzana",
    "Pera",
    "Limón",
    "Naranja"
)
```

```
Manzana
```

¿Tabulaciones o espacios?

> Nunca mezcles tabulaciones y espacios. El método de indentación más popular en Python es con espacios. El segundo más popular es con tabulaciones, sin mezclar unos con otros. Cualquier código indentado con una mezcla de espacios y tabulaciones debe ser convertido a espacios exclusivamente.
>
> Para proyectos nuevos, únicamente espacios es preferible y recomendado antes que tabulaciones. La mayoría de los editores presentan características para realizar esta tarea de manera sencilla.

Máxima longitud de las líneas

> Limita todas las líneas a un máximo de 79 caracteres.
>
> En el caso de largos bloques de texto (“docstrings” o comentarios), se recomienda limitarlos a 72 caracteres es.
>
> El método preferido para “cortar” líneas largas es utilizando la continuación implícita dentro de paréntesis, corchetes o llaves. Además, éstas pueden ser divididas en múltiples líneas envolviéndolas en paréntesis. Esto debe ser aplicado en preferencia a usar la barra invertida (“ ”).

Uso de barra invertida (“ ”)

La barra invertida aún puede ser apropiada en diversas ocasiones. Por ejemplo, en sentencias largas ó múltiples que no pueden utilizar continuación implícita, por lo que dicho carácter es aceptable:

[6]:
```
#archivo = open('ruta1/ruta2/ruta3/ruta4/archivo1.txt', 'w')
with open('ruta1/ruta2/ruta3/ruta4/archivo1.txt') as file_1, \
open('ruta1/ruta2/ruta3/ruta4/archivo2.txt', 'a') as file_2:
    var = file_1.read()
    print(var)
    file_2.write('\n' + var)
```

```
Texto en el archivo 1
```

En el caso de usar parentesis, corchetes o llaves el uso de la barra invertida es <b style=“color:red;>redundante, como en el siguiente ejemplo:

[7]:
```
mi_lista = [
    1, 2, 3, \
    4, 5, 6,
]

print(mi_lista)
```

```
[1, 2, 3, 4, 5, 6]
```

NOTA: Otro caso de esta índole es la sentencia assert.

OPERADORES BINARIOS

¿Debe romperse una línea antes o después de un operador binario?

Durante décadas, el estilo recomendado fue romper con los operadores binarios. Pero esto puede dañar la legibilidad, los operadores tienden a dispersarse en diferentes columnas de la pantalla y cada operador se aleja de su operando y pasa a la línea anterior. Aquí, el ojo tiene que hacer un trabajo adicional para saber qué elementos se agregan y cuáles se restan:

Forma errada

La forma correcta sería:

[8]:
```
valor1 = 3
valor2_promedio = 4
valor3 = 5
promedio_anterior = 2
valor4 = 4
valor_de_resultado_anterior = 2

resultado = (valor1
             + valor2_promedio
             + (valor3 - promedio_anterior)
             - valor4
```

```
            - valor_de_resultado_anterior
           )

print(resultado)
```

```
4
```

Líneas en blanco

Separar funciones de alto nivel y definiciones de clase con dos líneas en blanco.

Las definiciones de métodos dentro de una clase son separadas por una línea en blanco.

Se pueden utilizar grupos de líneasen blanco adicionales (escasamente) para separar grupos de funciones relacionadas.

Se puede usar líneas en blanco en funciones, (escasamente), para indicar secciones lógicas.

Importaciones

Las importaciones deben estar en líneas separadas, por ejemplo:

[9]:
```
import os
import sys
```

Sin embargo, es correcto decir:

[10]:
```
from subprocess import Popen, PIPE
```

Las importaciones deben estar agrupadas en el siguiente orden:

1) importaciones de la librería estándar
    1) importaciones terceras relacionadas
        3) importaciones locales de la aplicación / librería

Se debería poner una línea en blanco entre cada grupo.

Las especificaciones **all** van luego de las importaciones.

Las importaciones relativas (relative imports) para intra-paquete (intra-package) son altamente desalentadas (poco recomendadas) . Siempre usa la ruta absoluta del paquete para todas las importaciones. Incluso ahora que el PEP 328 está completamente implementado en Python 2.5, su estilo de importaciones relativas explícitas es activamente desalentado; Las importaciones absolutas (absolute imports) son más portables y legibles.

Al importar una clase desde un módulo que contiene una clase, generalmente está bien realizar esto:

from myclass import MyClass from foo.bar.yourclass import YourClass

Si esto causa coincidencias con nombres locales, realiza:

import myclass import foo.bar.yourclass

y usa:

“myclass.MyClass” “foo.bar.yourclass.YourClass”

Espacios en blanco en Expresiones y Sentencias

Manías

Evita usar espacios en blanco extraños en las siguientes situaciones:

Inmediatamente dentro de paréntesis, corchetes o llaves:

Sí: spam(ham[1], {eggs: 2}) No: spam(ham[1], { eggs: 2 })

Inmediatamente antes de una coma, un punto y coma o dos puntos:

Sí: if x == 4: print x, y; x, y = y, x No: if x == 4 : print x , y ; x , y = y , x

Sí: No:

Inmediatamente antes del paréntesis que comienza la lista de argumentos en la llamada a una función:

Sí: spam(1) No: spam (1)

Inmediatamente antes de un corchete que empieza una indexación o “slicing” (término utilizado tanto en el ámbito de habla inglesa como española):

Sí: dict[‘key’] = list[index] No: dict [‘key’] = list [index]

Más de un espacio alrededor de un operador de asignación (u otro) para alinearlo con otro:

Sí: x = 1 y = 2
long_variable = 3
No: x = 1 y = 2 long_variable = 3

Otras recomendaciones

Siempre rodea estos operadores binarios con un espacio en cada lado:

asignación (=),

asignación de aumentación (+=, -=,etc.),

comparaciones (==, <, >, !=, <>, <=, >=, in, not in, is,is not),

“Booleans” (and, or, not).

Si se utilizan operadores con prioridad diferente, considera agregar espacios alrededor del operador con la menor prioridad. Usa tu propio criterio, de todas menaras, nunca uses más de un espacio, y siempre mantén la misma cantidad en ambos lados.

Sí: i = i + 1 submitted += 1 x = x*2 - 1 hypot2 = x*x + y*y c = (a+b)* (a-b)
No: i=i+1 submitted +=1 x = x * 2 - 1 hypot2 = x * x + y * y c = (a + b) * (a - b)

No uses espacios alrededor del = (igual) cuando es utilizado para indicar un argumento en una función (“keyword argument”) o un parámetro con un valor por defecto.

Sí: def complex(real, imag=0.0): return magic(r=real, i=imag) No: def complex(real, imag = 0.0): return magic(r = real, i = imag)

Comentarios

Comentarios que contradigan el código es peor que no colocar comentarios. ¡Siempre realiza una prioridad de mantener los comentarios al día cuando el código cambie!

Los comentarios deben ser oraciones completas. Si un comentario es una frase u oración, su primera palabra debe comenzar con mayúscula, a menos que sea un identificador que comienza con minúscula. ¡Nunca cambies las mayúsculas/minúsculas de los identificadores! (Nombres de clases, objetos, funciones, etc.).

Si un comentario es corto, el punto al final puede omitirse.Comentarios en bloque generalmente consisten en uno o más párrafos compuestos por oraciones completas, por lo que cada una de ellas debe finalizar en un punto.

Deberías usar dos espacios luego de una oración que termine con un punto.

Comentarios en bloque

Los comentarios en bloque generalmente se aplican a algunos (o todos) códigos que los siguen, y están indentados al mismo nivel que ese código. Cada línea de un comentario en bloque comienza con un # (numeral) y un espacio (a menos que esté indentado dentro del mismo comentario).

Los párrafos dentro de un comentario en bloque están separados por una línea que contiene únicamente un # (numeral).

Comentarios en línea

(comentarios en la misma línea)

Usa comentarios en línea escasamente.

Un comentario en línea es aquel que se encuentra en la misma línea que una sentencia. Los comentarios en línea deberían estar separados por al menos dos espacios de la sentencia. Deberían empezar con un # (numeral) seguido de un espacio. Los comentarios en línea son innecesarios y de hecho molestos si su acción es obvia. No hagas esto:

x = x + 1 # Incrementar x

Pero algunas veces es útil:

x = x + 1 # Compensar el borde

Convenciones de nombramiento

Las convenciones de nombramiento (o nomenclatura) de la libreríaPython son un poco desastrosas, por lo que nunca vamos a obtener esto completamente consistente – sin embargo, aquí hay algunos de los actualmente recomendados estándares de nombramiento. Los nuevos módulos o paquetes (incluyendo frameworks de terceros) deberían estar escritos en base a estos estándares, pero donde una librería existente tenga un estilo diferente, la consistencia interna es preferible

Descriptivo: Estilos de nombramiento

No hay gran cantidad de diversos estilos de nombramiento. Te ayuda a ser capaz de reconocer qué estilo de nombramiento está siendo utilizado, independientemente de para qué están usados. Las tildes no deben incluirse, tómese como una regla para mantener la traducción con la correcta ortografía. En cualquiera de los casos, no deben utilizarse caracteres con tildes para el nombramiento.

Se distinguen los siguientes estilos:

b (una letra en minúscula)

B (una letra en mayúscula)

minúscula (lowercase)

minúscula_con_guiones_bajos (lower_case_with_underscores)

MAYÚSCULA (UPPERCASE)

MAYÚSCULA_CON_GUIONES_BAJOS (UPPER_CASE_WITH_UNDERSCORES)

PalabrasConMayúscula (CapitalizedWords) ("CapWords" o "CamelCase").

Nota: Al momento de usar abreviaciones en "CapWords", pon en mayúscula las letras de la abreviación. "HTTPServerError" es mejor que "HttpServerError".

minúsculaMayúscula (mixedCase) (difiere con "CapWords" por la primer letra en minúscula).

Palabras_Con_Mayúsculas_Con_Guiones_Bajos (Capitalizad_Words_With_Underscores) (¡horrible!).

Existe también el estilo de usar un prefijo único para identificar a un grupo de funciones relacionadas. Esto no es muy utilizado en Python, pero se menciona para cubrir los estilos en su totalidad. Por ejemplo, la función os.stat() retorna una tupla cuyos ítems tradicionalmente tienen nombres como st_mode, st_size, st_mtime y así. (Se realiza esto para enfatizar la correspondencia con los campos del "POSIX system call struct", que ayuda a los programadores a estar familiarizados.) La libraría X11 usa un prefijo "X" para todas sus funciones públicas. En Python, este estilo es generalmente innecesario ya que tanto los métodos como los atributos están precedidos con un objeto, al igual que los nombres de las funciones lo están con el nombre de un módulo.

> Además, la siguiente forma precedida por un guión bajo es reconocida (esta puede ser combinada con cualquiera de las convenciones nombradas anteriormente):

_simple_guion_bajo_como_prefijo (_single_leading_underscore): débil indicador de "uso interno". Por ejemplo, from M import * no importa los objetos cuyos nombres comienzan con un guión bajo.

simple_guion_bajo_como_sufijo_ (single_trailing_underscore_): utilizado como convención para evitar conflictos con un nombre ya existente, ejemplo: Tkinter.Toplevel(master, class_='ClassName')

___doble_guion_bajo_como_prefijo (___double_leading_underscore): al nombrar un atributo en una clase, se invoca el "name mangling" (dentro de la clase FooBar, ___boo se convierte en _FooBar___boo; véase abajo).

doble_guion_bajo_como_prefijo_y_sufijo (**double_leading_and_trailing_underscore**): los objetos y atributos que viven en "namespaces" controlados por el usuario. Ejemplo, **init**, **import** o **file**. Nunca inventes nombres como esos; únicamente utiliza losdocumentados.

Prescriptivo: Convenciones de nombramiento

Nombres para evitar:

Nunca uses los caracteres ‘l’ (letra ele en minúscula), ‘O’ (letra o mayúscula), o ‘I’ (letra i mayúscula) como simples caracteres para nombres de variables.

En algunas fuentes, estos caracteres son indistinguibles de los números uno y cero. Cuando se quiera usar ‘l’, en lugar usa ‘L’.

Nombres de paquetes y módulos

> Los módulos deben tener un nombre corto y en minúscula. Guiones bajos pueden utilizarse si mejora la legiblidad.
>
> Los paquetes en Python también deberían tener un nombre corto y en minúscula, aunque el uso de guiones bajos es desalentado (poco recomendado).
>
> Ya que los nombres de módulos están ligados a los de los archivos, y algunos sistemas operativos distinguen caracteres entre minúsculas y mayúsculas y truncan nombres largos, es importante que sean bastante cortos – esto no será un problema en Unix, pero podrá ser un problema cuando el código es portado a una antigua versión de Mac, Windows o DOS.
>
> Cuando un módulo escrito en C o C++ provee una interfaz de alto nivel escrita en Python, debe llevar un guión bajo como prefijo (por ejemplo, _socket).

Nombres de clases

> Casi sin excepción, los nombres de clases deben utilizar la convención “CapWords” (palabras que comienzan con mayúsculas).
>
> Clases para uso interno tienen un guión bajo como prefijo.

Nombres de excepciones

> Debido a que las excepciones deben ser clases, se aplica la convención anterior. De todas maneras, deberías usar un sufijo“Error” en los nombres de excepciones (en caso que corresponda a un error).

Nombres de variables globales

> (Esperemos que esas variables sean únicamente para uso dentro del módulo). Las convenciones son las mismas que se aplican para las funciones. Los módulos que estén diseñados para usarse vía:

from M import *

> deben usar el mecanismo **all** para prevenir la exportación de las variables globales, o usar la antigua convención especificando un guión bajo como prefijo (lo que tratarás de hacer es indicar esas globales como “no públicas”).

Nombres de funciones

> Las funciones deben ser en minúscula, con las palabras separadas por un guión bajo, aplicándose éstos tanto como sea necesario para mejorar la legibilidad. “mixedCase” (primera palabra en minúscula) es aceptado únicamente en contextos en donde éste es el estilo predominante (por ejemplo, threading.py) con el objetivo de mantener la compatibilidad con versiones anteriores.

Argumentos de funciones y métodos

Siempre usa self para el primer argumento de los métodos de instancia.

Siempre usa cls para el primer argumento de los métodos de clase.

Si los argumentos de una función coincide con una palabra reservada del lenguaje, generalmente es mejor agregar un guión bajo comosufijo antes de usar una abreviación u ortografía incorrecta. class_ es mejor que clss. (Tal vez es mejor evitar las coincidencias usando un sinónimo.)

Nombres de métodos y variables de instancia

Usa las mismas reglas que para el nombramiento de funciones: enminúscula con palabras separadas con guiones bajos, tantos como sea necesario para mejorar la legibilidad. Usa un solo guión bajo como prefijo para métodos no públicos y variables de instancia.

Constantes

Las constantes son generalmente definidas a nivel módulo, escritas con todas las letras en mayúscula y con guiones bajos separando palabras. Por ejemplo, MAX_OVERFLOW y TOTAL.

[ ]: