

Buenas prácticas en Python

PEP

- Python Enhancement Proposals
- <http://www.python.org/dev/peps/>
- De que tratan los PEP?
 - <http://www.python.org/dev/peps/pep-0001/>
- PEP8: Guia de estilo para código Python
 - Objetivo principal: legibilidad del código

Forma del código

Indentación

- Usar 4 espacios para definir un nivel de indentación, o usar tabs.
- No usar combinaciones de tabs y espacios.
- Definición e invocación de funciones con nombres largos o muchos caracteres, con parámetros alineados.

Yes:

```
# Aligned with opening delimiter
foo = long_function_name(var_one, var_two,
    ► var_three, var_four)

# More indentation included to distinguish this from the rest.
def long_function_name(
    var_one, var_two, var_three,
    ► var_four):
    print(var_one)
```

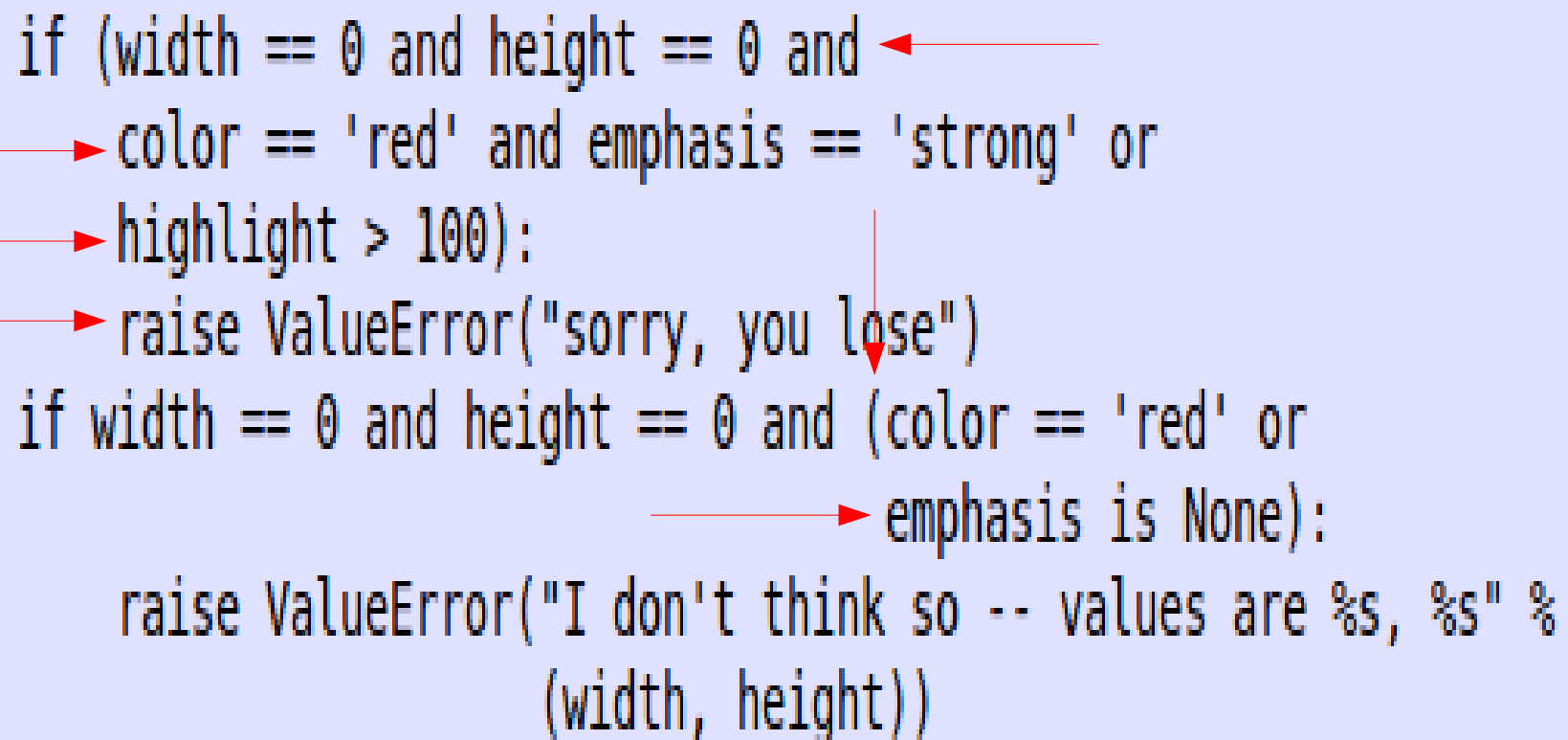
No:

```
# Arguments on first line forbidden when not using vertical alignment
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# Further indentation required as indentation is not distinguishable
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Tamaño de línea

- Tamaño máximo de la línea de código: 79 caracteres.
- Tamaño máximo de comentarios (línea o bloque) : 72 caracteres.
- Líneas de código largas (ej. condiciones complejas) deben partirse y agruparse con paréntesis.
- Al partir una línea con expresión compleja, hacerlo **después** del operador.



```
if (width == 0 and height == 0 and  
    color == 'red' and emphasis == 'strong' or  
    highlight > 100):  
    raise ValueError("sorry, you lose")  
if width == 0 and height == 0 and (color == 'red' or  
    emphasis is None):  
    raise ValueError("I don't think so -- values are %s, %s" %  
        (width, height))
```

The image shows a blue rectangular box with a white background, containing Python code. Red arrows point to various parts of the code: one points to the opening parenthesis of the first if statement, another points to the 'and' operator between 'height == 0' and 'color == 'red'', a third points to the 'or' operator between the first and second conditions of the first if statement, a fourth points to the 'highlight > 100' condition, a fifth points to the 'raise' keyword, a sixth points to the 'or' operator in the second if statement, and a seventh points to the 'emphasis is None' condition.

Líneas en blanco

- Separar definición de funciones y clases por 2 líneas en blanco.

```
10 def sumatoria(n1, n2, *resto):
11     resultado = n1 + n2
12     for x in resto:
13         resultado += x
14     return resultado
15
16
17 def imprimirN(cadena, n=10):
18     while n > 0:
19         print cadena
20         n -= 1
21
22
23 def multiplicar(x, y):
24     x = x * y
25     return x
26
27
```


Imports

- Un import por línea, o bien, múltiples funciones por import de módulo.
- Siempre poner imports al inicio del archivo, después de comentarios del módulo y antes e declaración de variables globales y constantes.
- Orden de los import:
 - Librerías estándar (ej. sys, random, math)
 - Librerías de terceros
 - Librerías locales

```
Yes: import os  
      import sys
```

```
No:  import sys, os
```

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

Espacios en blanco

- No ponerlos a lo interno de paréntesis.
- No ponerlos antes de : , ;
- No ponerlos antes del paréntesis o brackets que abre.
- No usar más de un espacio en blanco en asignación.

Yes: `spam(ham[1], {eggs: 2})`

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

Yes: `spam(1)`

No: `spam (1)`

Yes: `dict['key'] = list[index]`

No: `dict ['key'] = list [index]`

Yes: `if x == 4: print x, y; x, y = y, x`

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

Yes:

`x = 1`

`y = 2`

`long_variable = 3`

No:

`x = 1`

`y = 2`

`long_variable = 3`

Espacios en blanco

- En operadores binarios (como asignación, comparación, etc) usar un espacio a cada lado.
- En operadores con menor precedencia, se puede agrupar sin usar espacios.

Yes:

```
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)
```

Espacios en blanco

- No usar espacios en blanco con el = en parámetros con valores por defecto o con nombres clave:

Yes:

```
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)
```

Declaraciones compuestas

- Una línea por instrucción, no usar varias instrucciones en una línea.

Yes:

```
if foo == 'blah':  
    do_blah_thing()  
do_one()  
do_two()  
do_three()
```

Rather not:

```
if foo == 'blah': do_blah_thing()  
do_one(); do_two(); do_three()
```

Comentarios

- Siempre usar comentarios para describir funciones, módulos y partes del código complejas.
- Asegurarse de actualizar comentarios ante cambios en el código (no tener comentarios contradictorios).
- Siempre comenzar comentarios con mayúscula, a menos que inicien haciendo referencia a un nombre de variable.

Comentarios

- En comentarios de bloque, terminar oraciones con el punto.
- Comentarios de bloque se deben indentar al mismo nivel del código que comentan.
- Comentarios de línea deben ir en la misma línea de código que comentan, y separar el texto del # con un espacio en blanco.

```
x = x + 1
```

```
# Compensate for border
```

Docstrings

- Primer contenido dentro de funciones, módulos o clases.
- Todas las funciones a exportar en un módulo deben tener un docstring.
- Comienzan y terminan con `"""` (3 ")
- Primera línea es un resumen breve.
- Siguiendo líneas son descripción más detallada.

Docstrings

```
"""Return a foobang
```

```
Optional plotz says to frobnicate the bizbaz first.
```



```
"""
```

Convención de nombres

- No utilizar nombres de variables l (ele), O, I)i mayúscula).
- Nombres de módulos deben ser cortos y en minúscula.
- Nombres de clases deben ser en la forma CamelCase.
- Al crear Excepciones personalizadas deben usar la convención CamelCase y contener la palabra Error.

Convención de nombres

- Nombres de funciones deben ser en minúscula, separando palabras con `_`. NO usar mixedCase
- Los nombres de las constantes se definen en mayúscula y con `_` : `MAX_OVERFLOW`
- Las constantes se definen a nivel de módulo.

Recomendaciones generales

- Al comparar con None, nunca usar `==` o `!=`, sino `is` o `is not`.
- Al manejar excepciones, siempre mencionar excepciones específicas, no dejar el `except` solo. En casos extremos, usar `except Exception`.
- Al comparar tipos, usar función `isinstance`, ejemplo: `if isinstance(obj, int):`

Recomendaciones generales

- En secuencias (strings, listas, tuplas), aprovechar el hecho de que la secuencia vacía equivale a False

```
Yes: if not seq:  
      if seq:
```

```
No: if len(seq)  
     if not len(seq)
```

Recomendaciones generales

- No compare valores booleanos con True o False usando ==

Yes: `if greeting:`

No: `if greeting == True:`

Worse: `if greeting is True:`