Documentos y Estilo de Programación en `python`

Curso de Física Computacional - Guía de apoyo 2

M. en C. Gustavo Contreras Mayén.

1. Documentación del código

Es una buena costumbre documentar el código que vayamos creando, en parte por que se va a reutilizar y aunque nosotros seamos los usuarios, la mejor manera de ir perfeccionado las técnicas de programación, es a través de los comentarios y observaciones de otros usuarios.

Toma en cuenta que documentar el código, no es lo mismo que comentarlo, ya que los comentarios no se muestran como parte de la documentación que genera `python`.

Veremos que hay una manera de "leer" la documentación que se haya contenido dentro del código.

1.1. Convenciones para la documentación

La primer línea debe ser siempre un resumen corto y conciso del propósito del objeto.

Para ser breve, no se debe mencionar explícitamente el nombre o tipo del objeto, ya que estos están disponibles de otros modos (excepto si el nombre es un verbo que describe el funcionamiento de la función). Esta línea debe empezar con una letra mayúscula y terminar con un punto.

Si hay más líneas en la cadena de texto de documentación, la segunda línea debe estar en blanco, separando visualmente el resumen del resto de la descripción.

Las siguientes líneas deben ser uno o más párrafos describiendo las convenciones para llamar al objeto, efectos secundarios, etc.

Código 1: Ejemplo de documentación

```
1 def mi_funcion():
2     """
3     No hace mas que documentar la funcion.
4
5     No, de verdad. No hace nada.
6     """
7     pass
8
9 print(mi_funcion.__doc__)
```

Cuando ejecutamos la función `print` con el argumento en la terminal, veremos que se nos despliega el contenido dentro del bloque con las tres comillas, de esta manera le damos al usuario información al respecto de nuestro procedimiento.

2. Estilo de codificación

Cualquier lenguaje de programación cuenta con normas para la escritura de código, más allá del contexto de la lógica del programa, la propuesta de algoritmos, etc. siempre se debe de mantener un estándar en cuanto a la manera de escribir el código.

Esto facilita la lectura cuando nos proporcionan un código proveniente de otro autor, y por ello, es relevante e imprescindible que nos adaptemos a esos estándares de escritura de código.

A continuación revisaremos la guía de estilo vigente para el código con `python`.

2.1. Guías de estilo

Ahora que estás a punto de escribir códigos de `python` más largos y complejos, es un buen momento para hablar sobre estilo de codificación.

La mayoría de los lenguajes pueden ser escritos (o mejor dicho, formateados) con diferentes estilos; algunos son mas fáciles de leer que otros.

Hacer que tu código sea más fácil de leer por otros es siempre una buena idea, y adoptar un buen estilo de codificación ayuda tremendamente a lograrlo.

Para `python`, PEP 8 se erigió como la guía de estilo a la que más proyectos se adhirieron; promueve un estilo de codificación fácil de leer y visualmente agradable. Todos los que trabajan con `python` deben leerlo en algún momento.

2.2. Puntos importantes PEP 8

- Usar sangrías de 4 espacios, no tabs.
- Recortar las líneas para que no superen los 79 caracteres.
- Usar líneas en blanco para separar funciones y clases, y bloques grandes de código dentro de funciones.
- Cuando sea posible, poner comentarios en una sola línea.
- Usar docstrings.
- Usar espacios alrededor de operadores y luego de las comas, pero no directamente dentro de paréntesis: $a = f(1, 2) + g(3, 4)$.
- Nombrar las clases, módulos y funciones consistentemente; la convención es usar NotacionCamello (Camel Case) para clases y minusculas_con_guiones_bajos para funciones y métodos.
- No uses codificaciones estrafalarias si esperás usar el código en entornos internacionales. El default de `python`, UTF-8, o incluso ASCII plano funcionan bien en la mayoría de los casos.

- De la misma manera, no uses caracteres no-ASCII en los identificadores si hay incluso una posibilidad mínima de que gente que hable otro idioma tenga que leer o mantener el código.

2.3. Ejemplos del uso de PEP8

Espacios en blanco en expresiones y sentencias.

2.3.1. Evita espacios en blanco extra en las siguientes situaciones

Inmediatamente después de entrar en un paréntesis o antes de salir de un paréntesis, corchete o llave.

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

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

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

Inmediatamente antes de abrir un paréntesis para una lista de argumentos de una llamada a una función:

Si:	No:
`spam(1)`	`spam (1)`

Inmediatamente antes de abrir un paréntesis usado como índice o para particionar (slicing):

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

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

Si:	No:
`x = 1`	`x = 1`
`y = 2`	`y = 2`
`long_variable = 3`	`long_variable = 3`

2.4. Otras Recomendaciones

Rodea siempre los siguientes operadores binarios con un espacio en cada lado: asignación (=), asignación aumentada (+ =, − = etc.), comparación (==, <, >, ! =, <> , <=, >=, `in, not in, is, is not`), booleanos (`and, or, not`).

Usa espacios alrededor de los operadores aritméticos:

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

No uses espacios alrededor del signo "=" cuando se use para indicar el nombre de un argumento o el valor de un parámetro por defecto.

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

Generalmente se desaconsejan las sentencias compuestas (varias sentencias en la misma línea).

Si:	Preferiblemente no:

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

Aunque a veces es adecuado colocar un `if/for/while` con un cuerpo pequeño en la misma línea, nunca lo hagas para sentencias multi-claúsula.

Preferiblemente no:	Definitivamente no:

`if foo == 'blah': do_blah_thing()`
`for x in lst: total += x`
`while t <10: t = delay()`
	`if foo == 'blah': do_blah_thing()`
	`else: do_non_blah_thing()`
	`try: something()`
	`finally: cleanup()`
	`do_one(); do_two(); do_three(long, argument`
	`list, like, this)`
	`if foo == 'blah': one(); two(); three()`

Conforme vayas programando te darás cuenta de la ventaja de usar un estilo de programación adecuado, el uso de entornos de desarrollo, ahorra mucho tiempo para el formato del estilo, pero aún así, no debemos de confiarnos y dejar un código complicado de leer y entender.