NORMATIVA DE CÓDIGO
DESARROLLO DE SOFTWARE

MARZO 2023

María Ángela Romero Martin, Gonzalo Diez Villar, Ismael Plaza Martin
100472330, 100495825, 100499716
GRUPO 10
GRADO EN INGENIERÍA INFORMÁTICA

ÍNDICE DE CONTENIDOS

Este documento recoge todas las reglas que hemos modificado del código de convenciones estilísticas que describe 'Python PEP-8'. En él, hemos realizado un conjunto de recomendaciones con el objetivo de establecer una convención de programación que facilite la lectura del código.

1.REGLAS DE NOMBRADO

A continuación se describe todo lo relacionado con el convenio de las reglas de nombrado.

1.1 REGLAS DE NOMBRADO DE VARIABLES.

1.1.1 EMPLEAR PREFIJO INDICATIVO DEL TIPO DE VARIABLE.

Toda variable debe tener un prefijo que indique su correspondiente tipo de variable. Hemos decidido realizar esta modificación ya que en otros lenguajes de desarrollo a menudo es imperativo indicar el tipo de variable. Pensamos que indicar el tipo de variable no sólo ayudaría a mejorar la legibilidad del código, sino que, de cara al futuro, podría facilitar el trabajo de desarrolladores encargados del mantenimiento del código que hayan sido ajenos al proceso de elaboración del código.

A continuación se describe cómo se debe cumplir esta norma:

CORRECTO	INCORRECTO
`intA = 4`	`a = 4`
`floatB = 0.25`	`b = 0.25`
`strC = 'iHola Mundo!'`	`c = 'iHola Mundo!'`
`boolD = TRUE`	`d = True`

1.1.2 VARIABLES ESCRITAS EN EN LOWERCAMELCASE.

Todas las variables deberán estar escritas en 'lowerCamelCase'. Hemos decidido realizar esta modificación de estilo ya que en el resto de lenguajes de programación es más frecuente el uso de esta pauta de codificación, creando de esta forma cierta coherencia con otros idiomas informáticos.

A continuación se describe cómo se debe cumplir esta norma:

CORRECTO	INCORRECTO
`strFecha = ' 07-03-2020 '`	`fecha = ' 07-03-2020 '`
`strTexto = '¡Hola Mundo!'`	`texto = ' Hola Mundo '`
`intContadorHoras = 3`	`contador = 3`

1.2 REGLAS DE NOMBRADO DE FUNCIONES.

1.2.1 FUNCIONES ESCRITAS EN UPPERCASE.

Todas las funciones deberán estar escritas en 'UPPERCASE'. Hemos decidido tomar esta decisión ya que al tratar con un código muy extenso, será más fácil localizar una función de entre el resto del código.

A continuación se describe cómo se debe cumplir esta norma:

CORRECTO	INCORRECTO

`def CALCULARSUMA(lstNumeros):`
`  intSuma = 0`
`  for numero in lstNumeros:`
`  intSuma += numero`
`  return intSuma`
	`def HorasDelDia():`
	`  contador = 0`
	`  while contador < 24:`
	`  hora += 1`
	`  return contador`

`def ESTUDIANTES( strEstudiante ):`
`  lstEstudiantes = ('Paula', 'Jose', 'Enrique', 'Maria')`
`  if strEstudiante not in lstEstudiantes: print('No está')`
`  EXIT`
	`def horas_del_dia():`
	`  contador = 0`
	`  while contador < 24:`
	`  hora += 1`
	`  return contador`

`def SUMA(intA, intB):`
`  intResultado = intA + intB`
`  return intResultado`
	`def horasdeldia():`
	`  contador = 0`
	`  while contador < 24:`
	`  hora += 1`
	`  return contador`

1.3 REGLAS DE NOMBRADO DE CONSTANTES.

1.3.1 PREFIJO GUIÓN BAJO EN EL NOMBRADO DE CONSTANTES.

Todas las constantes deberán estar precedidas por un guión bajo al ser declaradas. Hemos decidido tomar esta decisión ya que de esta forma será más fácil distinguir una constante de una variable. Aparte del uso del guión bajo al comienzo, el nombre de la constante deberá estar escrito en 'lowerCamelCase' al igual que el resto de variables.

A continuación se describe cómo se debe cumplir esta norma:

CORRECTO	INCORRECTO
`_pi = 3.14159`	`PI = 3.14159`
`_fuerzaDeGravedad = 9.81`	`fuerza_de_gravedad = 9.81`
`_numeroEuler = 2.718281828459045`	`NumeroEuler = 2.718281828459045`

2. REGLAS DE ORGANIZACIÓN DE CÓDIGO

A continuación se describen todos los estándares relacionados con la organización de código.

2.1 RELATIVO A LOS ESTÁNDARES DE LÍNEAS.

2.1.1 LONGITUD DE LÍNEA.

Una línea de código tendrá como máximo una longitud de 120 caracteres. De acuerdo al estándar 'Python PEP-8', el máximo establecido son 79 caracteres, pero resulta más lógico y manejable establecer un máximo redondo de acuerdo a la base decimal. Por ello, hemos decidido elevar este número a 120 caracteres.

*2.1.2 LÍNEAS ENTRE DECLARACIONES DE IMPORTS.

Deberá haber una línea de separación entre la declaración de dos importes diferentes. De esta manera, será más fácil distinguir dos 'imports' diferentes e incluso se podrán añadir comentarios entre líneas explicando en qué consiste el módulo importado.

A continuación se describe cómo se debe cumplir esta norma:

CORRECTO	INCORRECTO

`import math`
 	
`import random`
 	
`import os`
	`import math`
	`import random`
	`import os`

`from contador.py import CONTADOR`
 	
`import datetime`
	`from contador.py import contador`
	`import datetime`

2.1.3 CAPACIDAD MÁXIMA DE LÍNEAS EN UN MÓDULO

Un módulo nunca deberá superar las 1000 líneas de código en su totalidad. De esta manera, se conforma un código más concreto y reducido, favoreciendo así la eficiencia de este y concienciando a los desarrolladores a no generar un código más extenso de lo estrictamente necesario. Además, establecer un límite de líneas propicia la división en módulos de un programa, lo cual de cara al futuro, contribuye a la depuración, reutilización y mantenimiento del programa.

2.1.4 CANTIDAD MÁXIMA DE ARGUMENTOS EN UNA FUNCIÓN.

Una función deberá recibir como máximo diez argumentos. Hemos decidido tomar esta decisión ya que diez argumentos parecen una cantidad generosa, y a partir de este número el código podría empezar a resultar desordenado y poco comprensible. De este modo, resultaría más eficiente modular dicha función en otras más pequeñas, lo que, al igual que la anterior regla, impulsa la depuración, reutilización y mantenimiento del programa.

2.1.5 DECLARACIÓN DE 'IF STATEMENTS' EN UNA SOLA LÍNEA.

Una sentencia de tipo 'if' podrá ser declarada en una sóla línea. Bajo este criterio, el código tendrá un aspecto menos dilatado y se reducirá el número de líneas que abarque un programa.

A continuación se describe cómo se debe cumplir esta norma:

CORRECTO	INCORRECTO
`if x < 10:print("x es menor que 10")`	`if x > 10:` `  print("x es mayor que 10")`

`if intNum not in lstNumbers:print('Not contained')`	`lista = ( 1,2,3 )`
	`if num not in lista:`
	`  print('Not contained')`

2.2 RELATIVO A LOS ESTÁNDARES DE ESPACIOS EN BLANCO.

2.2.1 NO DEBERÁ HABER ESPACIOS ANTES Y DESPUÉS DE LOS OPERADORES ARITMÉTICOS.

No deberá haber espacios ni antes ni después de un operador aritmético. Los operadores aritméticos incluyen los operadores de suma, resta, multiplicación, división, división entera, módulo y exponenciación. Al omitir el uso de un espacio, queda más evidente cuales son las variables que están siendo utilizadas en la operación, y resulta más sencilla la implementación del código. Esta norma no aplica al resto de operadores.

A continuación se describe cómo se debe cumplir esta norma:

CORRECTO	INCORRECTO
`print( intA+intB )`	`print(a + b )`
`print( intA-intB )`	`print(a - b)`
`print( intA*intB )`	`print(a * b)`
`print( intA/intB )`	`print(a / b)`
`print( intA//intB )`	`print(a // b)`
`print( intA**intB )`	`print(a ** b)`