Instituto Tecnológico de Costa Rica

Escuela de Computación
Curso: IC6831
Aseguramiento de la Calidad del Software
Grupo 2, II Semestre 2017
Profesor: Saúl Calderón Ramírez

# Avance 1, Proyecto Semestral
# Estándar de codificación

Integrantes:
Brandon Dinarte 2015088894
Armando López 2015125414
Andrey Mendoza 2015082908
Julian Salinas 2015114132

---

Domingo 10 de Setiembre del 2017
ITCR, Sede Central Cartago

# 1. Estándar de codificación

## 1.1. Introducción

"Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez". -Microsoft

El propósito de definir este estándar de códificacion es obtener un estilo de programación homogéneo que permita a todos los participantes afines comprender el sistema de software, y que en consecuencia pueda ser fácilmente mantenido. Entiéndase mantenibilidad como, la facilidad con que el sistema puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores o mejorar el rendimiento.

Adicionalmente, establecer el estándar permite una mayor eficiencia, pues los entornos de desarrollo integrado proveen las herramientas necesarias para cumplir con dicho estándar, impidiendo que el programador lo incumpla, y que, en consecuencia, no se dificulte la capacidad de comprensión, evitando posibles errores que deberían de depurarse en un futuro.

El proyecto se realizará en el lenguaje de programación Python, por tanto, es común que se pongan en práctica las propuestas de mejora del mismo (PEPs, Python Enhancement Proposals). Por esta razón, se tomará como base los estándares de *PEP 8* para el estilo de codificación y *PEP 256* como convención para la documentación interna del código fuente.

Podemos encontrar las referencias completas de dichos PEP en los siguientes enlaces:

- PEP 8: https://www.python.org/dev/peps/pep-0008/
- PEP 257: https://www.python.org/dev/peps/pep-0257/

Nota: Se asume que los lectores de este documento están familiarizados con los términos que en este se utilizan.

## 1.2. Herramientas para el cumplimento del estándar

Se hará uso de un ambiente de desarrollo integrado específico para el lenguaje de programación Python llamado PyCharm, este provee herramientas necesarias para cumplir con los estándares mencionados (PEP 8 y PEP 257). *PyCharm* analiza el código fuente de forma estática, haciendo revisiones básicas de módulos, clases, funciones y variables, así como, de documentación interna y

formato.

Específicamente, entre las principales inspecciones que realiza *PyCharm* se pueden mencionar:

- Estilo de código PEP 8
- Indentaciones incorrectas
- Funciones, clases o módulos obsoletos
- Código inalcanzable
- Argumentos incorrectos de una función
- Docstrings incorrectos o incompletos
- Paréntesis redundantes o incorrectos
- Declaraciones sin efecto
- Opacidad en los nombres
- Simplificación de verificación de variables booleanas
- Cláusulas de excepción en orden incorrecto
- Llamados a métodos de una clase que no corresponden
- Clases que deben implementar los métodos abstractos
- Comparaciones con None realizadas con operadores de igualdad
- Diccionarios que contiene claves repetidas
- Errores en operaciones de formato con cadenas
- Verificación de tipos al aplicar operadores

Además de estas, se pueden realizar otras inspecciones más específicas, las cuales pueden ser consultadas en la página oficial de *JetBrains*, distribuidor de *PyCharm*.

Para maximizar su utilidad se usará la extensión *SolarLint*, la cual proporciona de forma instantánea información útil sobre nuevos errores encontrados y problemas de calidad inyectados en el código fuente, lo cual, disminuye el tiempo que corresponde a las revisiones de la calidad del código.

Cabe destacar que es una herramienta versátil, ya que puede analizar código en diferentes lenguajes usando como base el server de *SonarQube*, esto permitirá que distintos módulos del sistema, sin importar el lenguaje usado, puedan ser

analizados.

La extensión *SolarLint* es de gran utilidad, sin embargo, no cubre ciertas características o reglas puesto que su objetivo no es ralentizar el IDE, por tanto, cada vez que se alcance un punto de control del sistema, este será analizado por medio de *SonarQube Scanner*, el cual requiere un poco más de tiempo tanto para inicializar como para realizar el análisis completo.

## 1.3. Reglas del estándar PEP 8

Las reglas del estándar se pueden encontrar en la página oficial de Python, sin embargo, se hará mención de las que son consideradas subjetivamente como las más importantes. Además, se establecerán reglas específicas donde el estándar de libertad o reglas opcionales.

- Indentación
  - Se debe indentar con 4 espacios en vez de usar tabuladores
  - Cuando una parte de una declaración if es suficientemente larga para requerir múltiples líneas, la clausulas se deben acomodar de tal manera que no se confunda con el resto del código (no se dice que exista una regla especifica en el estándar para esta situación)

- Tamaño máximo de línea
  - El límite es de 79 caracteres

- Líneas en blanco
  - Se deben dejar dos antes del inicio de una clase o una función de orden superior
  - La separación entre métodos de una clase debe ser de solo una línea
  - También se pueden usar para separar secciones lógicas de una función

- Codificación de archivos
  - Si se utiliza Python 2 con ASCII o Python 3 con UTF-8 no se debe declarar la codificación

- Importación de módulos
  - Cada importación de un módulo diferente debe realizarse en una línea aparte

- Siempre tiene que estar ubicados al inicio del archivo
- Deben estar agrupados en el siguiente orden: estándar, de terceros, locales

- Espacios en blanco en expresiones y declaraciones
  - Evitar inmediatamente dentro de paréntesis
  - Evitar entre una coma final y un paréntesis cercano
  - Evitar inmediatamente antes de una coma, un punto y coma o un doble punto (en este caso se hace excepción con el slice)
  - Evitar después del operador [=] de un parámetro por default
  - Se deben usar espacio entre operadores y sus operandos

- Comentarios
  - Capitalizar la primera letra
  - Conservar el nombre de identificadores dentro de los comentarios
  - Los comentarios en bloque deben estar indentados al mismo nivel que el código
  - Evitar comentarios en línea

- Convenciones de nombres
  - Los nombres que son visibles al usuario como partes públicas de un API deben reflejar más su uso que su implementación.
  - Variables y funciones: Se debe usar minúsculas y separar con guiones bajos cuando aplique
  - Clases: Debe iniciar con mayúscula en cada palabra del identificador. No usa guiones bajos para separar (CamelCase)
  - Constantes: Se debe usar mayúsculas, separadas por guiones bajos cuando aplique
  - Módulos: Deben tener nombres cortos, todo en minúsculas, separados con guiones bajos cuando aplique
  - Se deben evitar los identificadores [l], [I], [O], [o]

- Consistencia
  - Si bien el estándar brinda por defecto cierto nivel de consistencia, el programador, junto con su equipo, deben ser consistentes con los términos que utilizan

- El idioma oficial dentro del código será español, se usará nh cuando sea necesario escribir la ñ
- Python permite usar los caracteres [‘] y [“] para identificar cadenas; sin embargo, se utilizará solamente el carácter [“] para tal fin
- Si se debe hacer un salto de línea en una operación algebraica, el salto será antes del operador
- Los paréntesis o llaves de cierre en construcciones multilínea deben alinearse bajo el primer carácter del nombre de la variable

### 1.4. Reglas del estándar PEP 257

El objetivo de este PEP es estandarizar la estructura de alto nivel de los docstrings, que de forma simple esto es, qué cosas deberían contener dichos docstrings.

Con objetivos de consistencia se usarás triple dobles comillas [“””] para delimitar los docstrings. Además, debe haber al final de estos debe haber un salto de línea.

- *Docstrings* de una línea
  - Se usan únicamente para casos donde por ejemplo una función resulta en cosas obvias
  - No existen líneas en blanco antes o después del [“””]
  - Prescribe el efecto de la función o del método como un comando
- *Docstrings* de múltiples líneas
  - Consisten en un resumen de una línea donde empieza el [“””], seguido de un salto de línea y una descripción más elaborada.

El docstring para una función o método debe resumir su comportamiento y documentar sus argumentos, valor de retorno, efectos colaterales, posibles excepciones y restricciones de los contextos donde puede usarse (si es aplicable).

## Referencias


[1] Python Software Foundation, *Python Enhancement Proposals*, Recuperado de: `https://www.python.org/dev/peps/`

[2] Manuel Arias Calleja, *Estándares de codificación*, Recuperado de: `http://www.cisiad.uned.es/carmen/estilo-codificacion.pdf`

[3] Microsoft, *Revisiones de código y estándares de* , Recuperado de: https://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx?f=255&MSPPError=-2147217396