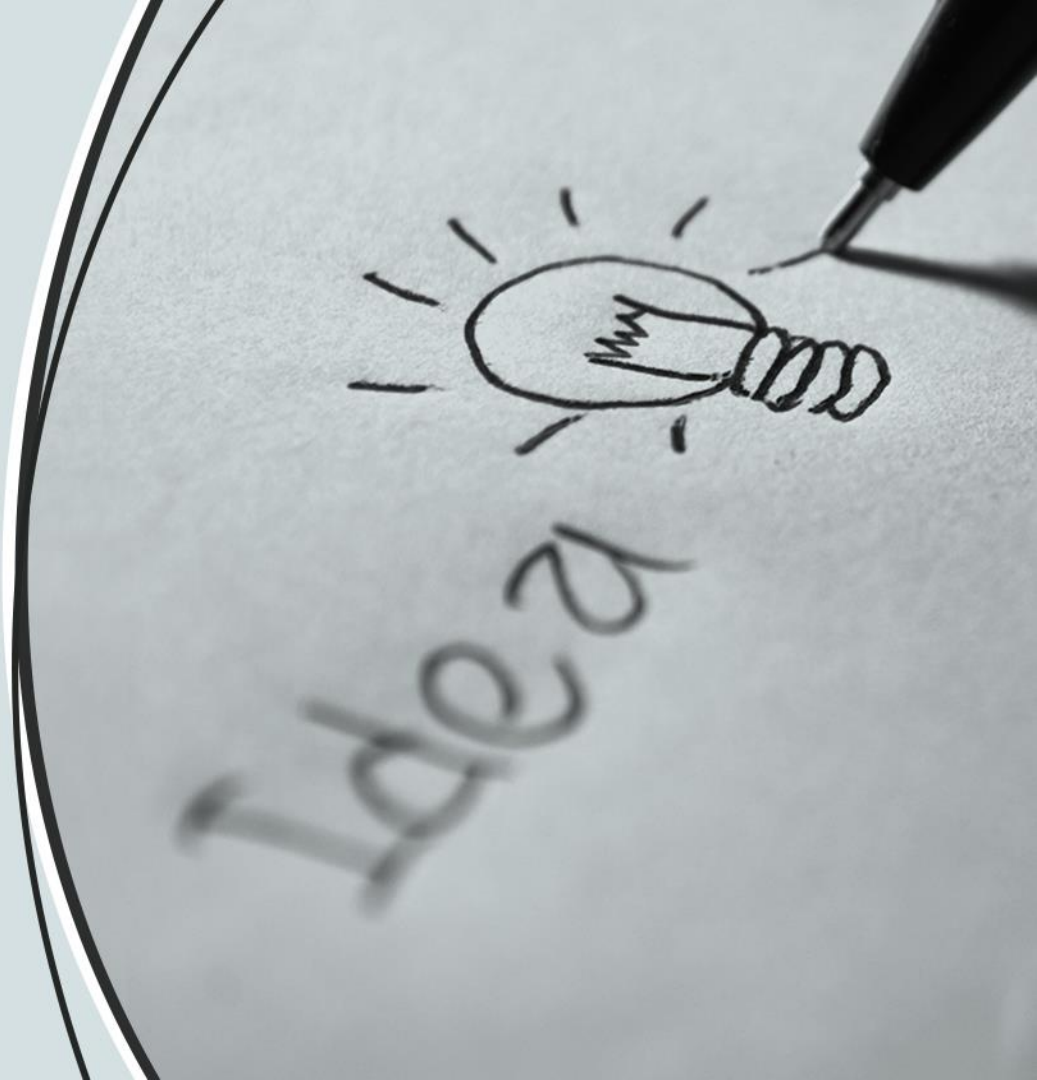
Documentación y Estilos de Programación

Herramientas Computacionales
Ing. Gustavo Andrés Salazar



Estilos de Programación

¿Qué son los estilos de programación?

El estilo de programación corresponde a un grupo de reglas o guías para escribir código.

El estilo de programación ayuda a que los programadores puedan leer y entender el código de manera más sencilla.

También ayuda al programador a evitar errores al momento de programar.

Estilos de Programación

Beneficios de los estilos de Programación

Los principales beneficios de utilizar los estilos de Programación son:

- Leer y entender el código más fácilmente.
- Evitar errores al momento de programar.
- Facilidad para mantener el programa.
- Reducción en el tiempo de Depuración.

Estilos de Programación

Características

Normalmente los estilos de programación están asociados al lenguaje de programación en el cual se esta desarrollando el programa, pero por lo general los estilos de programación abordan las siguientes características al momento de programar:

- Indentación (Indentation)
- Espacios en Blanco
- Corte de Líneas (Line Breaks)
- Variables
- Constantes
- Métodos / Funciones
- Clases

Estilos de Programación

Indentación

Esta característica corresponde a los espacios o tabs que se tienen desde el borde izquierdo de la hoja.

En algunos lenguajes de programación esta característica se utiliza delimitar bloques de código, en donde la indentación correcta es necesaria para el funcionamiento correcto y va más alla que un tema de estilo (Python). En otros lenguajes que utilizan llaves para delimitar bloques de código o sentencias, como por ejemplo C, C++ o Java es una característica de estilo.

Estilos de Programación

Indentación

Al momento de empezar un bloque de código o sentencia las siguientes líneas de código deben tener un nivel más de indentación.

Estilos de Programación

Indentación - Ejemplo

```
if (numero > 7 && valor < 10) { return numero + valor; }
else { return numero; }
```

```
if (numero > 7 && valor < 10)
{
return numero + valor;
}
else
{
return numero;
}
```

Estilos de Programación

Indentación - Ejemplo

```
if (numero > 7 && valor < 10)
{
    return numero + valor;
}
else
{
    return numero;
}
```

Estilos de Programación

Espacios en Blanco

El uso de espacios en blanco tanto de manera vertical como horizontal es importante al momento de escribir un programa.

Los espacios en blanco se deben tener al momento de definir los parámetros de una función, pasar los argumentos al momento de llamar una función y cuando se define una expresión.

Estilos de Programación

Espacios en Blanco

```
def prueba(valor1,valor2):
```

```
    res=valor1+valor2
```

```
prueba(1,2)
```

Estilos de Programación

Espacios en Blanco

```
def prueba(valor1, valor2):
```

```
    res = valor1 + valor2
```

```
prueba(1, 2)
```

Estilos de Programación

Espacios en Blanco

```
def func1():
    return 1

def prueba(valor1, valor2):
    res = valor1 + valor2 + func1()
    return res
```

Estilos de Programación

Espacios en Blanco

```
def func1():
    return 1


def prueba(valor1, valor2):
    res = valor1 + valor2 + func1()
    return res
```

Estilos de Programación

Corte de Líneas (Line Breaks)

Se recomienda no tener líneas de código con más de 80 caracteres.

Esto para evitar que la líneas sean demasiado largas al momento de leerlas y no sea necesario desplazar la ventana de manera horizontal.

Para esto algunos Editores/IDEs muestran una línea al momento de llegar a 80 caracteres.

Estilos de Programación

Corte de Líneas (Line Breaks)

```
def ingresar_registro(valor, registro, nombre, apellido, num_documento):
    return valor + registro + nombre + apellido + num_documento
```

Estilos de Programación

Corte de Líneas (Line Breaks)

```
def ingresar_registro(valor, registro, nombre,
                      apellido, num_documento):
```

Estilos de Programación

Variables

Los nombres de las variables deben ser descriptivos acorde a la información que van a almacenar.

Se deben definir en minúsculas con guiones bajos para separar palabras.

También se pueden nombrar con minúscula y Mayúscula la primera letra de la segunda palabra.

No usar como nombre de variables l (letra ele en minúscula), O (letra o mayúscula) o I (letra i mayúscula).

Estilos de Programación

Variables

```
preciofinal = 10
```

```
p = 10
```

Estilos de Programación

Variables

```
precio_final = 10
```

✓

```
precioFinal = 10
```

✓

Estilos de Programación

Constantes

Los nombres de las constantes se deben nombrar en mayúsculas (todos los caracteres).

Se recomienda usar guiones bajos para separar las palabras.

Estilos de Programación

Constantes

```
ERRORMESSAGE = "Error!"
```

```
errormessage = "Error!"
```

Estilos de Programación

Constantes

```
ERROR_MESSAGE = "Error!"
```

Estilos de Programación

Métodos / Funciones

Se debe manejar nombres relacionados con las acción o labor de la función o método.

Se debe nombrar en minúscula, con las palabras separadas por un guion bajo.

Estilos de Programación

Métodos / Funciones

```
def sumaValores(valor1, valor2):
    return valor1 + valor2
```

```
def SUMAVALORES(valor1, valor2):
    return valor1 + valor2
```

```
def SumaValores(valor1, valor2):
    return valor1 + valor2
```

Estilos de Programación

Métodos / Funciones

```
def suma_valores(valor1, valor2):
    return valor1 + valor2
```

✓

Estilos de Programación

Clases

Manejar nombres asociados al objeto que se va a definir.

Los nombres de las clases deben utilizar la convención “CapWords”, en donde cada palabra comienza en mayúscula y no se separan por guiones.

Estilos de Programación

Clases

```
class clientebanco:
```

```
class Clientebanco:
```

```
class cliente_banco:
```

Estilos de Programación

Clases

```
class ClienteBanco:
```

Documentación

¿Qué es la Documentación?

La documentación del código es un factor importante también al momento de desarrollar.

¡No todos pensamos de la misma manera!

A pesar que sea fácil de entender lo que hace es mejor explicar como se hace.

Documentación

Comentarios

Los comentarios en el código permiten que otros programadores puedan entender o saber que hace el código sin entrar en detalles del mismo.

De igual manera que se recomiendan unas buenas practicas al momento de documentar, también se tienen buenas practicas cuando se documenta.

Comentarios

Buenas Practicas

Comentarios que contradigan el código es peor que no colocar comentarios.

Si se actualiza el código se debe actualizar los comentarios.

Los comentarios deben ser oraciones completas.

Comentarios

Tipos de Comentarios

Los comentarios pueden ser categorizados en tres tipos:

Comentarios en Bloque
Comentarios en Línea
Cadenas de Documentación

Comentarios

Comentarios en Bloque

Este tipo de comentario es aquel que se aplica a un bloque de código, los cuales están indentados al mismo nivel del bloque.

```
numero = eval(input("Ingrese el número a dividir"))
divisor = eval(input("Ingrese el divisor"))
# Valido que el divisor no sea cero
if divisor != 0:
    res = numero / divisor
    print("El resultado es ", res)
else:
    print("El divisor no puede ser cero.")
```

Comentarios

Comentarios en Línea

Este tipo de comentario es aquel comentario que se encuentra en la misma línea de la instrucción.

Estos comentarios deben estar separados al menos por 2 espacios luego de la instrucción.

Se recomienda utilizarlos cuando realmente sean útiles y el comentario no sea obvio.

```
index += valor  # se ajusta el valor del arreglo
```

Comentarios

Cadenas de Documentación

Este tipo de comentario también son llamados "docstrings". Son utilizados para documentar módulos, funciones y clases.

En este caso es necesario detallar la información relevante de lo que se esta documentando.

La idea es poder escribir un texto descriptivo de la función, indicando las entradas (parámetros) y salida de la misma (return).

También se puede detallar las precondiciones y pos condiciones de la función.

Comentarios

Cadenas de Documentación

```
""" Función: validar_datos
Objetivo:
    La función valida que el identificador
    del producto pertenezca al tipo de
    producto ingresado.
Entradas:
    numero: identificador del producto
    tipo: tipo de producto
Salida (Boolean):
    True: Si el identificador
        pertenece al tipo de producto
    False: Si el identificador no
        pertenece al tipo de producto
"""
```

Tarea

Mapa Conceptual

A partir del articulo “I have to document THAT?”, realice un mapa conceptual en donde se detalle la importancia de la documentación en la computación.

https://www.ibm.com/developerworks/library/j-jtp0821/j-jtp0821-pdf.pdf

Plazo de Entrega: Lunes 6 de Marzo del 2017 antes de las 11:55 p.m.

Tarea Individual!

Se debe subir a la actividad de Blackboard.