Guía de estilo de programación en iMAT

En este documento se recoge una guía de estilo estándar (basado en la especificación de Python oficial PEP8) que se recomienda a los alumnos de iMAT a seguir durante todo el título.

Las buenas prácticas de escritura son importantes, no sólo por la evaluación en las asignaturas, sino también como programadores.

Fuente: https://peps.python.org/pep-0008/

Definición de variables, funciones, clases

Se utilizarán **nombres descriptivos** que ayuden a interpretar el código.

✕

```
b = b + p
```

☑

```
bote = bote + propina
```

Evidentemente, habrá ocasiones donde **x** e **y** serán unos magníficos nombres de variables cuando hablemos de unas coordenadas. :)

De todas las nomenclaturas disponibles, se seguirá como norma general **snake_case** para variables y funciones.

✕

```
IdAlumno = 202210101

def nombrelargofuncion(nombre, edad):
    print(nombre)
```

☑

```
id_aumno = 202210101

def nombre_largo_funcion(nombre, edad):
    print(nombre)
```

Las clases, sin embargo, se definirán siguiendo la nomenclatura **CamelCase**.

✕

```
class persona:

    def __init__(self, nombre):
        self.nombre = nombre
```

☑

```
class Persona:

    def __init__(self, nombre):
        self.nombre = nombre
```

Espacios en blanco

Cuando se trate de escribir expresiones matemáticas, se dejarán espacios en blanco entre las operaciones de menor prioridad. Por tanto, operaciones cómo la múltiplicación o división no tendrán espacios en blanco. Lo mismo ocurre con los paréntesis.

✕

```
x=y+1
x = y*2+1
x =(y+2)*3+ 1
```

☑

```
x = y + 1
x = y*2 + 1
x = (y+2)*3 + 1
```

Cuando se asigne un valor a una nueva variable, se hará de forma que se mantengan los espacios constantes en la asignación.

✕

```
bote    = 55
propina = 1
```

☑

```
bote = 55
propina = 1
```

Se evitará dejar espacios en blanco entre los paréntesis, corchetes o llaves.

✕

```
lista = [ 2 , 1 ]
diccionario = { 1 : "Luis",  2 : "Ana"}
print( lista )
print( diccionario )

def mostrar( nombre , edad ):
    print( nombre )
```

☑ Como se puede apreciar, siempre es recomendable dejar un espacio después de los separadores de elementos (en listas, diccionarios, argumentos, ...

```
lista = [2, 1]
diccionario = {1: "Luis", 2: "Ana"}
print(lista)
print(diccionario)

def mostrar(nombre, edad):
    print(nombre)
```

Definiciones multilínea

Cuando exista necesidad de tener más de una línea en la definición de un elemento, para mejor su comprensión...

✕

```
def mostrar(
    nombre, edad,
    direccion, telefono):
    print(nombre)

lista = [
    1, 2, 3,
    4, 5, 6
            ]
```

☑

```
def mostrar(
        nombre, edad,
        direccion, telefono):
    print(nombre)

lista = [
    1, 2, 3,
    4, 5, 6
    ]
```

Indentación

Aunque se recomienda identar con 4 espacios en lugar del tabulador, se puede utilizar éste último si alguien se encuentra más cómodo.

Lo que nunca hay que hacer es alternar y mucho menos tabular con un número distinto a 4 espacios.

✕

```
def mostrar(nombre):
  print(nombre)  # Identación con 2 espacios
```

☑

```
def mostrar(nombre):
    print(nombre)  # Identación con 4 espacios

def mostrar(nombre):
    print(nombre)  # Identación con 1 tab
```

Trabajando con booleanos

Se deben acostumbrar a trabajar con booleanos como valores de unas variables, pero también como el resultado de una condición.

✕

```
if error == True:
    print("Error")
```

☑

```
if error:
    print("Error")
```

Ancho del código fuente

Se intentará no trabajar con líneas my largas de código. Se recomienda no más de 79 caracteres por línea, pero se podría ampliar hasta 120 (máximo que soportan los notebooks de Jupyter, por ejemplo). Cuando nos encontremos con líneas más largas se buscará la forma de reducirlas. A continuación van unos ejemplos de mal ajuste de líneas.

✕

```
texto = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. In turpis nunc, scelerisque sed tellus eu,
cursus ultricies turpis. Integer metus tortor, tincidunt id mi sit amet, placerat gravida justo. Suspendisse
vel condimentum ex. Pellentesque viverra finibus felis sit amet convallis."

ingresos = (sueldo_bruto +
          intereses +
          (productividad - penalizaciones) -
          irpf -
          deducciones_ss)
```

☑

```
texto = """Lorem ipsum dolor sit amet, consectetur adipiscing elit.
           In turpis nunc, scelerisque sed tellus eu, cursus ultricies turpis.
           Integer metus tortor, tincidunt id mi sit amet, placerat gravida justo.
           Suspendisse vel condimentum ex. Pellentesque viverra finibus felis sit amet convallis."""


ingresos = (sueldo_bruto
            + intereses
            + (productividad - penalizaciones)
            - irpf
            - deducciones_ss)
```

Comentarios

Cuando se deban realizar **comentarios en una línea** se intentarán que estén sigan un criterio claro, que puede ser creando una separación 2 espacios respecto al código a comentar.

✕

```
import math

a = 7                    # Valor del primer cateto
b = 151       # Valor del segundo cateto

hipotenusa = math.sqrt((a**2) + (b**2))          # Se aplica el teorema de Pitágoras

print(f"Hipotenusa: {hipotenusa}")
```

☑

```
import math

a = 7  # Valor del primer cateto
b = 151  # Valor del primer cateto

h = math.sqrt((a**2) + (b**2))  # Se aplica el teorema de Pitágoras

print(f"Hipotenusa: {h}")
```

También se pueden crear comentarios de una línea para comentar un proceso posterior:

☑

```
# Librerías a utilizar
import math

# Inicialización
a = 7
b = 151

# Proceso
h = math.sqrt((a**2) + (b**2))

# Salida
print(f"Hipotenusa: {h}")
```

Los **comentarios multilínea** nos permiten ser más descriptivos :).

```
import math

"""
    |\
    | \
    |  \
  a |   \ h
    |    \
    |_ _ _\
       b
"""

a = 7  # Valor del primer cateto
b = 151  # Valor del primer cateto

h = math.sqrt((a**2) + (b**2))  # Se aplica el teorema de Pitágoras

print(f"Hipotenusa: {h}")
```

Documentación de código

Se recomienda, aunque puede utilizarse otro, utilizar el estilo **Google Docstring**.Lo importante es acostumbrarse a documentar el código.

```
import math

def calcular_hipotenusa(a, b):
    """Calcula la hipotenusa de un triángulo

    Args:
        a (int): Cateto #1
        b (int): Cateto #2

    Returns:
        h: valor de la hipotenusa

    Raises:
        excepcion: descripción

    Examples:
        Para los catetos a=2 y b=2 la hipotenusa vale 2.8
    """

    h = math.sqrt((a**2) + (b**2))  # Se aplica el teorema de Pitágoras
    return h
```

Importar librerías

Las librerías se importarán en las primeras líneas del código una a una.

✕

```
import pandas, numpy
```

☑

```
import pandas
import numpy

from numpy import array, random
```

Inicio del programa

Todos los programas deberán tener un punto de inicio claro de ejecución que vendrá dado por la función **main**.

```
if __name__ == "__main__":
    print("Inicio del programa")
    # Resto del código del programa
```

Tipificacion en funciones

Aunque va contra el espíritu libre de Python, los alumnos se deberán acostumbrar a definir el interfaz de las funciones en su propia definición para mejorar la calidad del software entregado.

```
In [3]:  def formatear(nombre: str, edad: int) -> str:
             return f"{nombre} ({edad})"

         formatear("Luis", 22)

Out[3]:  'Luis (22)'
```

Modularización del código

Los alumnos están acostumbrados a trabajar con módulos que les permitan mejorar la calidad del código desarrollado.

Por ejemplo, en 1º los alumnos están acostumbrados a trabajar con los siguientes módulos:

- main.py
- funciones.py
- objetos.py
- excepciones.py