Tabulaciones

Una **alternativa** a los **espacios** son las **tabulaciones**, pero como su tamaño **varía entre editores** y puede **afectar la legibilidad** o incluso dar **errores**, no se recomienda su uso.

Ademas en **Python 3** no está permitida la mezcla de espacios y tabuladores

```
def es_par(num):
     if num % 2 == 0
         return True
     return False
```

Cadenas de caracteres

Python soporta diferentes **dos** tipos de caracteres para crear **cadenas de caracteres**

Comillas simples o dobles: Para cadenas de caracteres de una línea o varias.

Triples comillas dobles: Para cadenas de documentación y bloques de cadenas de caracteres.

Docstring `" " " Sección de document ación con triples comillas dobles " " "`

Cadena de texto `print( " Hola, mundo ! ")`

Variables de módulo

Las variables de módulo definen ciertos aspectos generales:

__all__: Define qué objetos se importarán al hacer uso de import *.

__author__: Define el autor del módulo.

__version__: Define la versión del módulo actual.

Espacios en blanco

Los espacios en blanco en Python ayudan a mejorar la claridad del código, pero deben usarse con moderación.

Se debe añadir espacios entre elementos separados por comas, en ambos lados de una evaluación operativa y en asigna-ciones.

Hay que evitar espacios entre el nombre de funciones y su llamada, entre variables y el acceso interno con [], en operaciones lógicamente unidas y en paréntesis vacíos.

 # Uso correcto de espacios

```
resultado = (a + b, c - d)
```

Importaciones de código

Cada importación debe estar en una línea separada.

El orden es: librerías estándar, de terceros y luego las locales.

Se recomiendan importaciones absolutas.

Las importaciones relativas deben ser explícitas y breves.

Evitar el uso de * en importaciones.

```
import json
 import sys
from json import dump, load
```

Evitar hacer

```
from pkg import *
```

Longitud de líneas

La longitud de las líneas es un tema debatido que admite cierta flexibilidad.

Las longitudes idoneas son **79 caracteres por línea**, **72 para comentarios**, y **99 si el equipo acuerda usar líneas más largas**.

El propósito es tener una mejor lectura del **código** sin que tenga que haber saltos de linea.

Posición de los operadores

Cuando se tienen muchos operadores en nuevas líneas, las líneas deben de comenzar con el operador.

```
suma_total = (a
        + b )
```

Posición de líneas en blanco

En **Python**, el uso de **líneas** en **blanco** es **clave** para **mejorar la organización** y **legibi-lidad** del **código**.

2 líneas en blanco: rodean funciones y definiciones de clases de nivel principal.

1 línea en blanco: entre la definición de métodos dentro de una clase.

1 línea adicional: entre bloques lógicos de código para mayor claridad.

1 línea en blanco al final del archivo: todo archivo con contenido debe terminar con una línea en blanco.

```
class MiClase:

  def metodo_1(self):


def funcion_principal():
```

Saltos de linea

Cuando se pretenda hacer un salto de línea se utiliza el carácter \

```
if kg > 93 and dias_h ast a_v er ano < 10\
and destino == 'Playa':()
```