PYTHON INTERMEDIO

¿El zen de python?

20 principios de software (1999) a la hora de escribir código.

Bello es mejor es que feo: Código limpio y estético.

Explicito es mejor que implícito: Ser más claros al momento de codificar las cosas para que otros puedan entender lo que estamos haciendo.

Simple es mejor que complejo: Irse por el camino más fácil posible no tirar mucho código si no es necesario.

Complejo es mejor que complicado: Si tenemos que tirar más líneas para que el programa se entienda, tirelas papa.

Plano es mejor que anidado: Ojo con las funciones anidadas (una depende de la otra) es mejor no tanta anidación.

Espaciado es mejor que denso:

Legibilidad: Que el código sea comprensible.

Los casos especiales no son lo suficientemente especiales como para romper las reglas (sin embargo, la practicidad le gana a la pureza): Respetar las reglas estéticas que brinda Python siempre y cuando no se pierda la legibilidad del código, si esto llega a pasar siempre elegir el camino de la legibilidad.

Los errores no pasan silenciosos: Maneje los errores mijo, a menos de que este error sea silenciado.

Frente a la ambigüedad, no adivine: Si un algoritmo cambia de significado dependiendo del contexto debemos de repensar la solución.

Debería haber una, y preferiblemente sola, una manera obvia de hacerlo. (A pesar de que esa manera no sea obvia a menos que seas holandés): Dar soluciones precisas a los problemas o errores que presenta nuestro código.

Ahora es mejor que nunca: Implementar la solución a los problemas en el momento inmediato en el que se pueda.

A pesar de que nunca es muchas veces mejor que *ahora* mismo: Si en el inmediato momento planteamos la solución de un error con malas practicas mejor deje para después

Si la implementación es difícil de explicar, es una mala idea, y si es fácil de explicar, es una buena idea

¡Los espacios de nombres son una gran idea!: (namespaces)

¿Entornos virtuales?

Estos son subcopias de versiones posteriores o la versión actual de un lenguaje de programación “x”

¿Para qué? Pues bueno, estos sirven para que todas las funcionalidades de nuestro proyecto que tiene un entorno virtual no se dañen si hay una actualización del lenguaje de programación.

En resumidas cuentas, son un lenguaje de programación “x” separado para un proyecto con sus propios módulos.

Creación de un entorno virtual:

Comando crear un entorno virtual:

py -m venv “nombre del entorno virtual”

-m : Indica que vamos a modificar el funcionamiento original por otra cosa; -m significa modulo en py.

venv: Modulo virtual enviroment. Vamos a llamar al módulo venv

Comando activar/entrar al entorno virtual:

Se sugiere crear el alias “avenv” para el comando

Linux: source venv/bin/activate

Windows: .\venv\Scripts\activate

```
(venv) C:\Users\Admin\Desktop\ejemplo>
```

Si se activó bien el entorno virtual saldrá un ven junto con la ruta del proyecto

Para salir/desactivar el entorno virtual:

deactivate

Instalación de dependencias:

Para la instalación de módulos que no vienen por defecto con Python se usa el instalador de paquetes “pip”

Para instalar dependencias en un entorno virtual es necesario primero entrar en dicho entorno.

pip freeze: Comando para ver que módulos hay instalados en este momento y sus versiones.

pip freeze >> requirements.txt: Comando para crear un txt con las librerías que tenemos instaladas en nuestro venv.

pip install -r “nombre de archivo con librerías”: Comando para instalar las librerías de un venv remoto.

Errores en el código:

Hay dos principales tipos de errores dentro de Python, los errores de sintaxis (SyntaxError) y las excepciones (Exception)

Errores de sintaxis: Se denominan typo, al detectarlo Python para la ejecución del programa

Excepciones: Existen varias algunas de las más comunes son:

- KeyBoardInterrupt: Se paro el programa por medio del teclado
- KeyError: No existe la llave a la que intentas acceder en el diccionario
- IndexError: No existe el índice de la lista al que intentas acceder
- FileNotFoundError: No se encuentra el archivo que deseas abrir
- ZeroDivsionError: No se puede dividir por cero
- ImportError: Error al importar un modulo

Lo mejor es que al chocar con un error buscar desde lo ultimo hasta lo primero del código.

Python al toparse con un error lo eleva, lo lleva a la función principal del programa.