¿Cuáles son los tipos de Datos en Python?

En Python los tipos de datos básicos serían:

- **Booleanos**, representa un valor de verdad, puede ser True (verdadero), o False (falso)
- **Números,** pueden ser una gran variedad de elementos diferentes, desde enteros estándar (es decir, 1, 2, 3, etc.), hasta números decimales grandes, fracciones, "float" , etc.
- **Cadenas de texto**, Strings (str), cadenas de caracteres, tienen que ir entre comillas simples o dobles ('Hola mundo'),
- **Bytes and byte arrays,** que se utilizan para manejar datos binarios.
- **None**, Tipo especial que representa la ausencia de valor.
- **Listas**, secuencias ordenadas y mutables de elementos (por ejemplo, [1, 2, 3], ['a', 'b', 'c']).
- **Tuplas**, secuencias ordenadas e inmutables de elementos (por ejemplo, (1, 2, 3), ('a', 'b', 'c')).
- **Sets,** Conjuntos, colecciones no ordenadas de elementos únicos (por ejemplo, {1, 2, 3}, {'a', 'b', 'c'}).
- **Diccionarios**, (Dictionaries), colecciones no ordenadas de pares clave-valor (por ejemplo, {'nombre': 'Juan', 'edad': 30}).

Aunque, no es necesario declarar explícitamente el tipo de dato de una variable en Python; el lenguaje lo inferirá basándose en el valor que se le asigna.

¿Qué tipo de convención de nomenclatura deberíamos utilizar para las variables en Python?

En Python, se recomienda seguir las PEP 8, que son las guías de estilo para el código Python, es una guía de estilo extensa que entre muchas más indicaciones, nos guiará en las convenciones a la hora de poner nombre en variables, clases, en cómo tabular o usar sangrías, etc … por ejemplo, podemos encontrar el siguiente tipo de sugerencias a la hora de nombrar variables:

- Snake case para variables y funciones: En snake case, todas las letras son minúsculas y las palabras están separadas por guiones bajos. Por ejemplo: nombre_de_usuario

- Pascal case para clases: En Pascal case, cada palabra comienza con mayúscula y no hay separadores entre palabras. Por ejemplo: MiClase, Persona, ClienteNuevo.

- Evitar nombres de una sola letra, a menos que se use en contextos muy locales: Es preferible utilizar nombres descriptivos que indiquen el propósito de la variable o función.

- Usar nombres descriptivos y significativos: Los nombres de variables deben reflejar claramente el propósito o la función de la variable en el contexto de tu programa. Por ejemplo, en lugar de x, puedes usar contador, suma_total, etc.

¿Qué es un Heredoc en Python?

En programación a las cadenas de texto multilínea se les suele llamar "Heredoc". Al poner texto con varias líneas en Python pueden darse errores puesto que el fin de la línea es para Python el fin de la sentencia, y puede retornar un error, la forma más sencilla de solucionarlo es usando triples comillas, y da igual que sean dobles o simples, tanto para abrir el texto como para cerrarlo.

Ejemplo:

texto_varias_lineas= """La forma más fácil

de poner varias líneas

es con triples comillas"""

¿Qué es una interpolación de cadenas?

La interpolación de cadenas es un proceso mediante el cual se insertan valores de variables o expresiones dentro de una cadena de texto. Este proceso permite construir cadenas de manera dinámica, combinando texto estático con valores variables. Hay varias formas de hacerlo, por ejemplo:

Usando el método **`format()`** en una cadena y pasar los valores a ser interpolados como argumentos al método.

Ejemplo:

```
nombre='Ana'
edad=52
secuencia='Hola, mi nombre es {} y tengo {} años.' .format(nombre,edad)
print(secuencia)
```

Usando **f-strings**: A partir de Python 3.6, puedes utilizar f-strings, que son cadenas de texto precedidas por la letra **f**, donde las expresiones entre llaves **`{}`** son evaluadas y su resultado se inserta en la cadena.

Ejemplo:

```
nombre="Ana"
edad=52
secuencia=f"Hola, mi nombre es {nombre} y tengo {edad} años."
print(secuencia)
```

¿Cuándo deberíamos usar comentarios en Python?

En Python, los comentarios son útiles para proporcionar explicaciones, aclaraciones o documentación dentro del código, deben ser claros, concisos y estar actualizados. Los comentarios mal escritos o desactualizados pueden ser engañosos y dificultar la comprensión del código en lugar de facilitarla.

Ayudan a organizar el código, explicar el propósito de una sección (por ejemplo, que empieza la zona de la barra de navegación en un texto HTML), sirven para aclarar partes complicadas o difíciles de entender, pueden también indicar tareas que quedan pendientes de desarrollar...

Se aconseja no abusar de los comentarios dentro del código e intentar usar nombres de variables, clase, métodos que sean autoexplicativos.

En Python, los comentarios se indican utilizando el símbolo #. Todo lo que sigue al símbolo # en una línea se considera un comentario y es ignorado por el intérprete de Python.

¿Cuáles son las diferencias entre aplicaciones monolíticas y de microservicios?

Las aplicaciones monolíticas y las de microservicios son dos arquitecturas de software distintas que se utilizan para desarrollar aplicaciones. La elección entre una y otra depende de las necesidades específicas del proyecto y del equipo de desarrollo.

Aquí hay algunas diferencias clave entre ambas:

- **Monolítica**: En una aplicación monolítica, toda la funcionalidad de la aplicación se desarrolla y despliega como una sola unidad. La escalabilidad en una aplicación monolítica puede ser más difícil, ya que debe escalarse en su totalidad. El mantenimiento más sencillo, ya que solo hay una unidad que desarrollar, sin embargo, los cambios en una parte de la aplicación pueden afectar a otras partes, lo que puede hacer que sea más arriesgado. Es posible que varios equipos trabajen en diferentes partes de la aplicación, pero todos comparten el mismo código base. Esto puede llevar a conflictos.

- **Microservicios**: En una arquitectura de microservicios, la aplicación se divide en pequeños servicios independientes que se pueden desarrollar, desplegar y escalar de forma independiente. Cada servicio realiza una función específica y se comunica con otros servicios a través de interfaces bien definidas. Cada servicio se puede escalar de forma independiente según sea necesario. Esto permite un uso más eficiente de los recursos. El desarrollo y mantenimiento pueden ser más complejos, ya que hay múltiples servicios que deben coordinarse. Sin embargo, los cambios en un servicio no afectan a otros servicios, lo que permite una mayor independencia, En una arquitectura de microservicios, cada servicio puede ser desarrollado por equipos independientes, lo que permite una mayor una mayor autonomía para cada equipo.