# La precedencia de operadores en el lenguaje.

Python aplica los operadores según un orden determinado por su prioridad. A continuación, se muestran los operadores de más prioritarios (en la parte superior de la tabla) a menos prioritarios (en la parte inferior de la tabla):

| **Operador** | **Descripción** |
|---|---|
| (expresiones...) [expresiones...], {clave: valor...} {expresiones...} | Visualización o unión de tuplas, listas, diccionarios o conjuntos |
| x[índice], x[índice:índice], x(argumentos...), x.atributo | Extracción de datos de estructuras, referencia a atributos |
| await x | Expresión await |
| ** | Exponenciación |
| +x, -x, ~x | Especificación de signo, operador "no" a nivel de bits |
| *, @, /, //, % | Multiplicación, multiplicación de matrices, división, "floor division", resto |
| +, - | Adición y sustracción |
| <<, >> | Desplazamiento a nivel de bits |
| & | "y" a nivel de bits |
| ^ | "xor" a nivel de bits |
\| | "o" a nivel de bits |
| in, not in, is, is not, <, <=, >, >=, ==, != | Comparaciones, identificación y pertenencia |
| not x | "no" booleano |
| and | "y" booleano |
| or | "o" booleano |
| if - else | Expresión condicional |
| lambda | Expresión lambda |

Como se ha comentado, éste es el orden por defecto, pero siempre podemos utilizar paréntesis para determinar el orden en el que queremos que se apliquen los operadores.

## Diferencia entre los operadores en Python

### Comparador de igualdad (==)

El operador == se utiliza para comparar que dos elementos son iguales. Esto es, los objetos de las variables que se comparan tienen el mismo contenido, aunque puede ser diferentes objetos.

### Comparador de identidad (is)

El operador ìs se utiliza para comparar la identidad de dos objetos. Esto es, si los objetos a los que las variables hacen referencia son el mismo o no. Lo indica que esta comparación es más estricta que la realizada por el operador ==.

El operador is devuelve verdadero cuando los dos objetos a comparar son el mismo. Por otro lado, el operador == devuelve verdadero si el contenido es el mismo, pero no necesariamente el objeto. Cuando hablamos de datos primitivos, como los enteros, reales o cadenas de texto el comportamiento de los dos operadores es indistinguible.

## Try-Catch

```
try:
        # Código a ejecutar
        # Pero podría haber errores en este bloque

except <tipo de error>:
        # Haz esto para manejar la excepción
        # El bloque except se ejecutará si el bloque try lanza un
      error

else:
        # Esto se ejecutará si el bloque try se ejecuta sin errores

finally:
        # Este bloque se ejecutara siempre
```

- EL bloque try es el bloque con las sentencias que quieres ejecutar. Sin embargo, podrían llegar a haber errores de ejecución y el bloque se dejará de ejecutarse.
- El bloque except se ejecutará cuando el bloque try falle debido a un error. Este bloque contiene sentencias que generalmente nos dan un contexto de lo que salió mal en el bloque try.
- Siempre deberías de mencionar el tipo de error que se espera, como una excepción dentro del bloque except dentro de <tipo de error> como lo muestra el ejemplo anterior.
- Podrías usar except sin especificar el <tipo de error>. Pero no es una práctica recomendable, ya que no estarás al tanto de los tipos de errores que puedan ocurrir.
- EL bloque try es el bloque con las sentencias que quieres ejecutar. Sin embargo, podrían llegar a haber errores de ejecución y el bloque se dejará de ejecutarse.
- El bloque except se ejecutará cuando el bloque try falle debido a un error. Este bloque contiene sentencias que generalmente nos dan un contexto de lo que salió mal en el bloque try.
- Siempre deberías de mencionar el tipo de error que se espera, como una excepción dentro del bloque except dentro de <tipo de error> como lo muestra el ejemplo anterior.
- Podrías usar except sin especificar el <tipo de error>. Pero no es una práctica recomendable, ya que no estarás al tanto de los tipos de errores que puedan ocurrir.