Operador	Descripción	Ejemplo
+	Suma	`>>> 3 + 2` `5`
-	Resta	`>>> 4 - 7` `-3`
-	Negación	`>>> -7` `-7`
*	Multiplicación	`>>> 2 * 6` `12`
**	Exponente	`>>> 2 ** 6` `64`
/	División	`>>> 3.5 / 2` `1.75`
//	División entera	`>>> 3.5 // 2` `1.0`
%	Módulo	`>>> 7 % 2` `1`

Vide

Esto es como hacer una funcion

```
def TuFuncion(TuParametro1, TuParametro2):
    print("funcion")
    return x

TuFuncion(TuParametro1, TuParametro2) #y Se ejecuta
```

Esto va en todos los Archivos

```
if __name__ == "__main__":
    #Código
```

Esto lo que hace es que el nombre debe ser main para ejecutarse, y como siempre es main el código siempre se va a ejecutar en este caso llamamos a las funciones

Métodos

```
len(Tu_lista, elementos) -> Te da la cantidad de elementos de tu lista
lista.append(x) -> Agrega elementos o variables a tu lista
```

1. Funciones de conversión y manipulación de tipos:

- `abs(x)`: Devuelve el valor absoluto de un número.
- `all(iterable)`: Retorna `True` si todos los elementos de un iterable son verdaderos.
- `any(iterable)`: Retorna `True` si al menos un elemento de un iterable es verdadero.
- `ascii(object)`: Devuelve una representación legible de un objeto, escapando los caracteres no ASCII.
- `bin(x)`: Convierte un número entero a su representación binaria.
- `bool([x])`: Convierte un valor a `True` o `False`.
- `bytearray([source[, encoding[, errors]]])`: Devuelve un arreglo de bytes modificable.
- `bytes([source[, encoding[, errors]]])`: Devuelve un arreglo de bytes inmutable.
- `chr(i)`: Devuelve el carácter Unicode correspondiente a un entero.
- `complex([real[, imag]])`: Crea un número complejo.
- `dict([**kwargs])`: Crea un diccionario.
- `float([x])`: Convierte un número o cadena a un número flotante.
- `format(value[, format_spec])`: Formatea un valor de acuerdo con `format_spec`.
- `frozenset([iterable])`: Devuelve un conjunto inmutable.
- `hex(x)`: Convierte un número entero a su representación hexadecimal.
- `int([x[, base]])`: Convierte un número o cadena a un entero.
- `len(s)`: Devuelve la longitud de un objeto (cadenas, listas, etc.).
- `list([iterable])`: Crea una lista.
- `oct(x)`: Convierte un número entero a su representación octal.
- `ord(c)`: Devuelve el valor Unicode de un carácter.
- `repr(object)`: Devuelve una representación imprimible de un objeto.
- `round(number[, ndigits])`: Redondea un número al número de dígitos decimales especificado.
- `set([iterable])`: Crea un conjunto.
- `slice(stop)`: Devuelve un objeto slice que puede ser usado para cortar secuencias.
- `str([object])`: Convierte un objeto a una cadena.
- `tuple([iterable])`: Crea una tupla.
- `type(object)`: Devuelve el tipo del objeto.

- `zip(*iterables)`: Toma múltiples iterables y los agrupa en tuplas.

2. Funciones de entrada/salida:

- `input([prompt])`: Toma entrada del usuario como una cadena.
- `print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`: Imprime objetos en la salida estándar.
- `open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)`: Abre un archivo y devuelve un objeto de archivo.

3. Funciones de operaciones matemáticas:

- `divmod(a, b)`: Devuelve el cociente y el resto como una tupla.
- `max(iterable, *[, key, default])`: Devuelve el valor máximo de un iterable.
- `min(iterable, *[, key, default])`: Devuelve el valor mínimo de un iterable.
- `pow(x, y[, z])`: Devuelve `x` elevado a la potencia `y`, y opcionalmente módulo `z`.
- `sum(iterable[, start])`: Devuelve la suma de un iterable.

4. Funciones de iteración y filtrado:

- `filter(function, iterable)`: Filtra los elementos de un iterable usando una función.
- `map(function, iterable, ...)`: Aplica una función a todos los elementos de un iterable.
- `range([start], stop[, step])`: Genera una secuencia de números.
- `reversed(seq)`: Devuelve un iterador que recorre la secuencia en sentido inverso.
- `sorted(iterable, *, key=None, reverse=False)`: Devuelve una lista ordenada de los elementos en un iterable.

5. Funciones de manejo de excepciones:

- `isinstance(object, classinfo)`: Verifica si un objeto es una instancia de una clase o una tupla de clases.

- `issubclass(class, classinfo)`: Verifica si una clase es una subclase de otra.

6. Funciones relacionadas con el ámbito y el entorno:

- `globals()`: Devuelve un diccionario representando el espacio de nombres global.
- `locals()`: Devuelve un diccionario representando el espacio de nombres local.
- `eval(expression[, globals[, locals]])`: Evalúa una expresión dentro de los espacios de nombres opcionales.
- `exec(object[, globals[, locals]])`: Ejecuta un bloque de código dentro de los espacios de nombres opcionales.
- `dir([object])`: Devuelve una lista de nombres en el ámbito local o los atributos de un objeto.

7. Funciones para trabajar con módulos y atributos:

- `getattr(object, name[, default])`: Devuelve el valor del atributo llamado `name` de un objeto.
- `hasattr(object, name)`: Verifica si un objeto tiene un atributo con el nombre especificado.
- `setattr(object, name, value)`: Establece el valor del atributo de un objeto.
- `delattr(object, name)`: Elimina el atributo de un objeto.

8. Funciones para trabajar con objetos y clases:

- `callable(object)`: Verifica si un objeto es invocable (si es una función, por ejemplo).
- `id(object)`: Devuelve el identificador único de un objeto.
- `super([type[, object-or-type]])`: Devuelve un objeto proxy que delega llamadas a una clase padre.

9. Funciones para trabajar con memoria y objetos complejos:

- `memoryview(obj)`: Devuelve una vista de la memoria del objeto.
- `object()`: Devuelve una nueva instancia de la clase más básica.

10. Funciones útiles para la depuración y el desarrollo:

- `help([object])`: Muestra la ayuda para un objeto o módulo.
- `__import__(name, globals=None, locals=None, fromlist=(), level=0)`: Esta es la función utilizada por `import` internamente.