1. OPERACIONES CON DATOS:

Según el tipo/resultado operación: () paréntesis para agrupar, y precedencia de operadores.

PEP 8: operadores Siempre colocar un espacio en blanco, antes y después de un operador

1.1. ASIGNACIÓN.

La operación de asignación, permite además de asignar un "valor" a una variable, declararla.

Se utiliza el símbolo **=** para asignaciones.

¿*FORMATOS DE ASIGNACIÓN*?

Python, dispone de varias formas de asignar:

- **Simple**:

  Variable **=** valor

- **Múltiple:**(n = n)

  Es necesario que haya, el mismo número de variables que valores a recibir.

```
var1 [, var2 [, var3] ] . . . = valor1 [, varlor2 [, valor3] ] . . .
```

¿*Y si alguno no es N-N*?

Entonces da error,a no ser que….

☑ Completar con explicacion.

1.2. ARITMÉTICAS:

- Las operaciones aritméticas básicas:

Unarias:

- Negación `a = -5` a es `-5`

Binarias o más:

+ Suma `a = 10 + 5` a es `15`

- Resta `a = 12 - 7` a es `5`

***** Multiplicación `a = 7 * 5` a es `35`

****** Exponente `a = 2 ** 3` a es `8`

/ División simple `a = 12.5 / 2` a es `6.25`

→Devuelve **siempre** un número de coma flotante.

// División entera `a = 12.5 / 2` a es `6.0`

→Devuelve la **parte entera** (cociente), descartando la parte fraccional.

% Módulo `a = 27 % 4` a es `3`

→(Realiza una division entera) Devuelve el **resto**, en la división entera.

Notas:

+ Las divisiones por cero producen un error.
+ Operaciones con algún operando float, devuelve float, pues se convertirán los enteros a flotantes:

1.3. ARITMÉTICAS CON CADENAS, TUPLAS Y LISTAS.

- Para cadena, tuplas y listas.

+ Concatenación:

→Une cadenas, también concatena tuplas, listas, pero siempre que sean del mismo tipo/clase.

***** "multiplicación" Copia:

→ Repite la cadena tantas veces como se indique.

1.4. ASIGNACIÓN + OPERADORES (ARITMÉTICOS Y BINARIOS):

● Permiten, que se pueda realizar una operación y asignar el resultado.

+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
***=**	x *= 3	x = x * 3
****=**	x **= 3	x = x ** 3
/=	x /= 3	x = x / 3
//=	x //= 3	x = x // 3
%=	x %= 3	x = x % 3

● Otras con operadores de numeros binarios (operadores bitwise)

&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

1.5. OPERADORES DE NÚMEROS BINARIOS (OPERADORES BITWISE):

● Los operadores bitwise se utilizan para comparar números (binarios):

&	AND	Establece cada bit a 1 si ambos bits son 1.
|	OR	Establece cada bit a 1 si uno de dos bits es 1.
^	XOR	Establece cada bit a 1 si solo uno de dos bits es 1.
<<	Cero rellenar a la izquierda	Desplazar a la izquierda presionando ceros desde la derecha y dejar que los bits más a la izquierda caigan-
>>	Desplazar a la derecha	Desplácese a la derecha empujando copias de la parte más a la izquierda desde la izquierda, y deje que los bits más a la derecha se caen.
~	NOT	Invierte todos los bits.

1.6. RELACIÓN / COMPARACIÓN.

● Devolverán un valor booleano: True o False.

==	Igual que	`5 == 7`	`False`
!=	Distinto que	`rojo != verde`	`True`
<	Menor que	`8 < 12`	`True`
>	Mayor que	`12 > 7`	`True`
<=	Menor o igual que	`12 <= 12`	`True`
>=	Mayor o igual que	`4 >= 5`	`False`

Nota: Si comparamos tipos diferentes, obtendremos un error.

● **Identidad estricta**:

Los operadores de identidad estricta **`is`** y **`is not`** se utilizan para comparar no solo que dos objetos sean iguales, sino que **sean el mismo objeto**, es decir que apunten a la misma ubicación de memoria:

`is` El operador is / *is*., devuelve True si tienen el mismo id.

`is not` Este operador es el formato negado de is, devuelve True si no tienen el mismo id.

1.7. LÓGICOS → CONDICIÓN COMPUESTA

Y para evaluar más de una condición simultáneamente, se utilizan **operadores lógicos**:

Devolverán un valor booleano: True o False.

not not /*not*. . No lógico

```
not(x < 5 and x < 10)
```

and and / *and*. Y lógico

```
5 == 7 and 7 < 12       False and False  False
9 < 12 and 12 > 7       True and True    True
9 < 12 and 12 > 15      True and False   False
```

or or / *or*.. O lógico

```
12 == 12 or 15 < 7      True or False    True
7 > 5 or 9 < 12         True or True     True
```

xor / . xor lógico

```
4 == 4 xor 9 > 3        True o True      False
4 == 4 xor 9 < 3        True o False     True
```

- **Pertenencia** (Membresia-condición de miembro de una entidad):

Los operadores de membresía se utilizan para probar si una secuencia se presenta en un objeto:

in El operador in / *in*. Se utiliza para verificar si un valor está presente en una lista, tupla, etc.

1.8. PRECEDENCIA DE OPERADORES:

Como en otros lenguajes.