Clase 3

**Operadores y definición de Expresiones en Python**

**Operadores**

**¿Qué son?**

Formalmente, los operadores son aplicaciones, cálculos que se llevan a cabo sobre dos argumentos conocidos como operandos.

Operando [operador] Operando

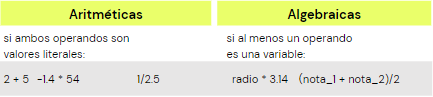
- / \* +

**EXPRESIONES**

Se denomina expresión al conjunto que forman los operandos y la operación.

Sumar, restar, dividir o multiplicar, tienen algo en común, y es que sus operadores son operadores aritméticos que sirven para trabajar con números.

Los operadores aritméticos (+, -, /, \*) dan lugar a expresiones de distintos tipos:



**El tipo lógico**

**Tipos de datos**

Los números, imágenes, textos, y sonidos, si algo tienen en común es que podemos percibirlos como información, pero hay un tipo de dato distinto, más básico.

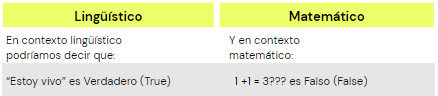
Es tan básico, que quizás cueste entenderlo como un tipo de dato.

Y ese, es el tipo lógico.

El tipo lógico es el tipo de dato más básico de la información racional, y representa únicamente dos posibilidades:

* Verdadero
* Falso

También denominamos a este tipo como Booleano o Binario.



**Negación**

Si negamos una cosa que es verdad, esta se convierte en mentira. Por lo tanto, si negamos una cosa que es mentira, esta se convierte en verdad.

* No Verdadero = Falso
* No Falso = Verdadero

¿Y en la programación?

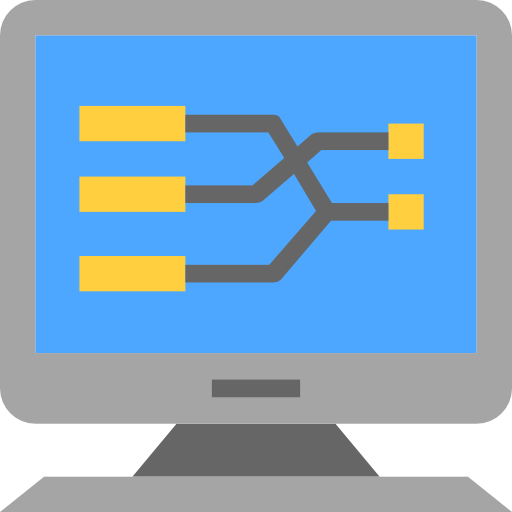
Por ejemplo, a un ordenador podemos preguntarle cosas matemáticas

**>>> 1 + 1 == 3 False**

Aquí estamos preguntando si al sumar 1 con 1 el resultado es 3 y Python ya sabe decirnos que esto es falso (false)

Y si le preguntamos si 1 + 1 es igual a 2?

**>>> 1 + 1 == 2 True**

**Operadores relacionales**

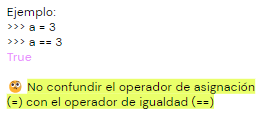
En programación, los operadores relacionales son símbolos que se usan para comparar dos valores.

Si el resultado de la comparación es correcto, la expresión es considerada verdadera (True), y en caso contrario será falsa (False).

**Igualdad**

El operador de igualdad sirve para preguntarle a nuestro programa si ambos operandos son iguales.

Devolverá True si son iguales, y False si son distintos. Este operador se escribe con dos signos igual (==).



**Desigualdad**

El operador de Desigualdad sirve para preguntarle a nuestro programa si ambos operandos son distintos.

Devolverá True si son distintos, y False si son iguales.

Ejemplo:

>>> a = 3

>>> a != 3

False

Este operador se escribe como un signo de exclamación y un signo igual (!=) como tachando al operador de igualdad.

**Menor que**

El operador Menor que sirve para preguntarle a nuestro programa si el primer operando es menor que el segundo operando.

Ejemplo:

>>> 7 < 3

False

>>> 1 < 15

True

Devolverá True si el primero es menor al segundo, y False si el primero es mayor que el segundo. Este operador se escribe con un signo de menor que (<).

**Menor Igual que**

El operador Menor igual que sirve para preguntarle a nuestro programa si el primer operando es menor que el segundo operando o si ambos son iguales.

Ejemplo:

>>> 7 <= 3

False

>>> 15 <= 15

True

Devolverá True si el primero es menor o igual al segundo, y False si el primero es mayor que el segundo.

Este operador se escribe con un signo de menor que y un igual (<=).

**Mayor que**

El operador Mayor que sirve para preguntarle a nuestro programa si el primer operando es mayor que el segundo operando.

Ejemplo:

>>> 7 > 3

True

>>> 1 > 15

False

Devolverá True si el primero es mayor al segundo, y False si el primero es menor que el segundo. Este operador se escribe con un signo de mayor que (>).

**Mayor igual que**

El operador Mayor igual que sirve para preguntarle a nuestro programa si el primer operando es mayor que el segundo operando, o si ambos son iguales.

Ejemplo:

>>> 7 >= 3

True

>>> 15 >= 15

True

Devolverá True si el primero es mayor o igual al segundo, y False si el primero es menor que el segundo.

Este operador se escribe con un signo de mayor que y un igual (>=).

**¿Operadores en Strings?**

No solo podemos hacer operaciones relacionales en números, también podemos hacerlas en strings.

Ejemplo:

>>> 'Hola' == 'Hola’

True

>>> a = 'Hola’

>>> a[0] != 'H'

False

🧐 También podemos comparar en Listas, Booleanos y más tipos de datos.

**Tipo Lógico**

Los Booleanos tienen un valor aritmético por defecto. True tiene un valor de 1 y mientras tanto False tiene un valor de 0. Es decir, tienen un valor binario que se utiliza para poder operar entre sí.

Ejemplo:

>>> True > False

True

>>> True \* 3

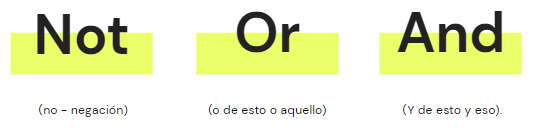
3

>>> False / 5

0.0

**Operadores lógicos**

Existen varios tipos de operadores lógicos en Python. Pero nos estaremos enfocando en los tres más básicos y utilizados:



**Not**

El not es la negación o también conocida como el NO. Es un poco especial, ya que solo afecta a los tipos lógicos True y False; solo requiere un operando en una expresión.

Ejemplo:

>>> not True

False

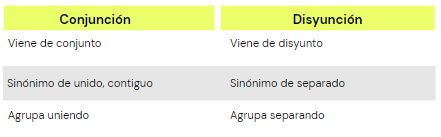
>>> not True == False

False

* Negación Lógica (NO)
* Solo afecta a los lógicos

**Más operadores**

Los operadores lógicos nos permiten crear grandes expresiones, estos operadores se presentan en dos formas:



**And**

El operador de conjunción, es decir, el que agrupa a través de la unión, es el operador lógico AND, en castellano conocido como Y.

Pero, ¿qué es lo que une? Este operador une una o varias sentencias lógicas:

* Estoy vivo **y** estoy dando un curso.

Ambas sentencias están unidas por un Y y ambas son afirmaciones verdaderas. Y, ¿visto en conjunto?

* VERDADERO y VERDADERO

Si tenemos dos afirmaciones que son verdaderas, evidentemente estaremos diciendo la verdad. Python también puede comprender esto, es decir, si preguntamos sobre dos afirmaciones unidas por un Y, sabrá decir si es verdadero o falso.

Ejemplo:

>>> 2 > 1 and 5 > 2

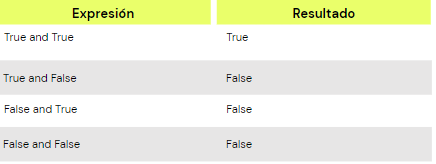
True

>>> 5 > 20 and 20 < 1

False

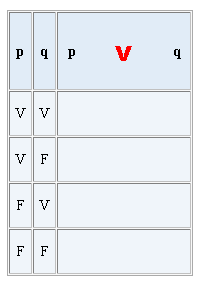
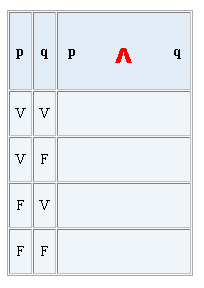
Para Python, una unión and es solamente verdadera(True) cuando, y solo cuando, toda la sentencia o conjunto de afirmaciones es verdadera. Es decir, cuando las dos afirmaciones son verdaderas. Si yo tengo una afirmación verdadera y otra falsa, Python siempre va a tomar como que esto es falso(False) si usamos el operador and.

**And**

****

**Tabla de verdad del And**

La cantidad de combinaciones entre dos proposiciones lógicas unidas por un and son cuatro y ya las hemos analizado en la tabla anterior. Eso se llama tabla de verdad



**Or**

Ahora, veamos el operador de disyunción denominado Or en castellano O.

Si el AND unía, el OR separa. Es decir, si a Python le pregunto por dos afirmaciones, y al menos una es (verdadera)True, Python me dirá que esta afirmación es True.

Ejemplo:

>>> 2 > 1 or 5 > 2

True

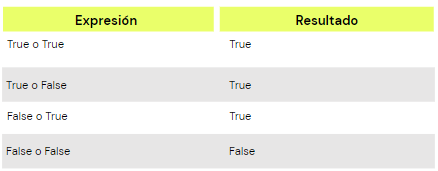
>>> 5 < 20 or 20 < 1

True

Para Python, una separación or es solamente verdadera(True) cuando, y sólo cuando, una de las sentencias o conjuntos de afirmaciones es True, es decir, cuando yo tengo una afirmación verdadera.

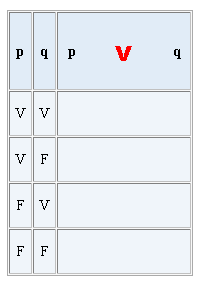
Si yo tengo una afirmación verdadera y otra falsa, python siempre va a tomar como que esto es True si usamos el operador Or.

**Or**

****

**Tabla de verdad del Or**

Su tabla de verdad quedaría así:



**Expresiones anidadas**

Hemos visto que existen un montón de expresiones distintas y como pueden suponer, es posible crear combinaciones entre estas.

A esto, se lo denomina expresiones anidadas.

El problema es que se pueden definir grandes expresiones con multitud de operadores y operandos, y si no sabemos como Python las interpreta a la hora de resolverlas, podríamos causar algunos errores sin querer.

**Normas de precedencia**

Ya que equivocarse es el pan de cada día, usaremos esta sección para poder aprender las normas de precedencia y aprenderemos como Python resuelve las expresiones complejas con los distintos tipos de operadores.

Si recuerdan, en la clase 1 vimos las procedencias de operadores numéricos:

* Términos entre paréntesis.
* Potenciación y raíces.
* Multiplicación y división.
* Suma y resta.

**Normas de precedencia**

Nos sirven para cuando tengamos que trabajar con expresiones anidadas que sean demasiado grandes.

Ejemplo:

>>> a = 15

>>> b = 12

>>> a \*\* b / 3\*\*a / a \* b >= 15 and not (a%b\*\*2) != 0

False

Nota: En la práctica nunca, o casi nunca, trabajaríamos con una expresión de este estilo, es por mero ejemplo.

**¿Por qué nos dio False?**

* Expresiones de cualquier tipo entre paréntesis.
* Expresiones aritméticas por sus propias reglas.
* Expresiones relacionales de izquierda a derecha.
* Operadores lógicos (not tiene prioridad, ya que afecta al operando).

**Operadores de asignación**

En esta sección vamos a ver unos tipos de operadores aritméticos que actúan directamente sobre la variable actual modificando su valor. Es decir, no necesitan dos operandos, solamente necesitan una variable numérica.

Por eso, se les llama operadores de asignación. El operador de asignación más utilizado y el cual hemos utilizado hasta ahora es el signo = . Este operador asigna un valor a una variable: número = 15

Además de este operador, existen otros operadores de asignación compuestos, que realizan una operación aritmética básica sobre la variable.

**Suma en asignación**

Teniendo ya declarada una variable, podemos directamente sumarle un valor,

Por ejemplo 1:

>>> a = 0

>>> a += 1

1

Ahora, cada vez que yo haga a+=1 se incrementará el valor de a en 1. Para poder aplicar cualquier operador en asignación se debe tener una variable previamente declarada, de lo contrario nos devolverá un error

Mostrar ejemplos de suma en asignación

**Resta en asignación**

También podemos directamente restarle un valor

Por ejemplo 5:

>>> a = 50

>>> a -= 5

45

Ahora, cada vez que yo haga a-=5 a se disminuirá el valor de a en 5

Mostrar ejemplos de resta en asignación

**Producto en asignación**

También podemos directamente hacer un producto a un valor.

Por ejemplo 10:

>>> a = 5

>>> a \*= 10

50

Ahora, cada vez que hagamos a\*=10 se multiplicará el valor de a en 10

Mostrar ejemplos de producto en asignación

**División en asignación**

También podemos directamente hacer una división a un valor.

Por ejemplo 2:

>>> a = 10

>>> a /= 2

5

Ahora, cada vez que hagamos a/=2 se dividirá el valor de a en 2

**Módulo en asignación**

También podemos directamente hacer un módulo a un valor.

Por ejemplo 2:

>>> a = 10

>>> a %= 2

0

Ahora, cada vez que hagamos a%=2 se hará el módulo de a en 2

Mostrar ejemplos de módulo en asignación

**Potencia en asignación**

También podemos directamente hacer una potencia a un valor.

Por ejemplo 2:

>>> a = 5

>>> a \*\*= 2

25

Ahora, cada vez que hagamos a\*\*=2 se hará una potencia de a en 2

**OPERADORES DE ASIGNACIÓN**

