El futuro digital es de todos

MinTIC

Universidad Industrial de Santander

MisiónTIC 2022

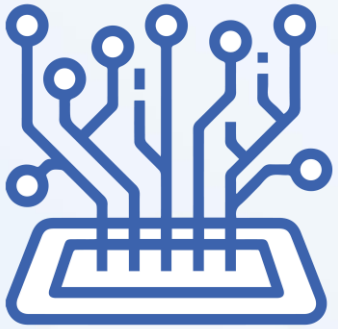
FUNDAMENTOS DEL LENGUAJE PYTHON

TEMA 5: OPERADORES ARITMÉTICOS BÁSICOS

5.1. Introducción

Bienvenidos nuevamente a este curso de Python. En la presente sección, operadores aritméticos y lógicos, veremos todos los operadores básicos que nos ofrece Python a la hora de programar, con el fin de aprender a hacer, desde operaciones sencillas tales como la suma y la resta, hasta operaciones mucho más complejas, donde haya que hacer uso de todo lo que aprenderemos.

El contenido de esta sección abarcará los siguientes temas: operadores aritméticos, operadores de comparación, operadores lógicos y, finalmente, la precedencia de los operadores. Algunos de estos temas pueden sonar un poco intimidantes, pero… ¡Es todo lo contrario, son extremadamente útiles! Veremos cada uno de los operadores con mucho detalle durante esta sección, de tal forma, que no vaya a quedar ninguna duda, no sin antes hacer una breve introducción de cada uno de estos temas a continuación.



5.1.1. Operadores aritméticos

Los operadores aritméticos son la base de todo cálculo que se pretenda realizar en cualquier lenguaje de programación, ya sean estos sencillos o complejos. Los operadores más básicos consisten en la suma, la resta, la multiplicación y división, en últimas, los que todos usamos en nuestro día a día. Python, evidentemente, incluye dichos operadores, además, tiene unos adicionales que realizan operaciones un poco más complejas, estos operadores son la potenciación, el módulo y la división entera, de todos ellos hablaremos con muchísimo más detalle un poco más adelante.

5.1.2. Operadores de comparación

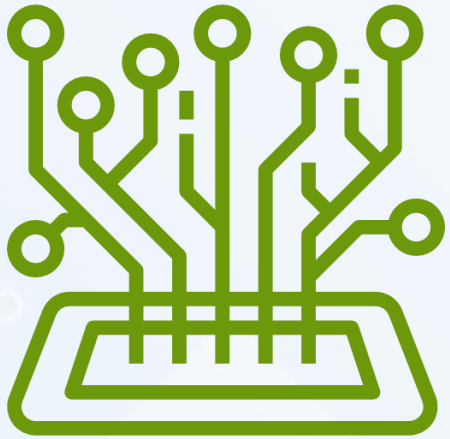
Los operadores de comparación sirven, como su nombre lo indica, para comparar entre dos datos, con el fin de que nos dé una respuesta que nos pueda ser útil. Un ejemplo de operador de comparación es la igualdad, con esta podemos ver si un dato es igual a otro.

5.1.3. Operadores Lógicos

Los operadores lógicos suenan un poco más intimidantes, ¿a que sí? Pero no te preocupes, que realmente no son muy complejos y, no obstante, son extremadamente útiles para realizar diversas tareas,, como veremos llegado el momento. Por ahora, quédate con la idea de que sirven para determinar y realizar condiciones.

5.1.4. Precedencia de operadores

Tal como en las matemáticas, los operadores se realizan siguiendo un orden específico y estándar para todo lenguaje de programación. Si no recuerdas cuál es el orden de las operaciones matemáticas no te preocupes, que, de una forma muy sencilla, lo recordarás.



5.2. Operadores aritméticos

Como se había mencionado en la introducción de la sección, los operadores aritméticos son la base de todo cálculo matemático que se pretenda realizar en todo lenguaje de programación, es decir, Python no es la excepción. Abordaremos cada uno de los operadores aritméticos que nos ofrece Python por orden de dificultad, empezando desde los más sencillos hasta los más difíciles, contando con ejemplos para cada uno. Es importante destacar que, para realizar una operación aritmética, los valores que se vayan a operar deben ser ambos del mismo tipo. Sin más preámbulos, comenzamos.

5.2.1. Suma

El primer operador aritmético que aprenderemos es la suma. La suma en todo lenguaje de programación de alto nivel consiste en hacer una adición que uno le indique a un número o una variable. La sintaxis en la que se realiza una suma en Python es muy sencilla, basta con poner un número o una variable, seguido del símbolo suma (+) y otro número o variable, tal como se escribiría una suma en una línea de cuaderno. En el ejemplo en pantalla le pedimos a Python que imprima el resultado de 10 más 20. Recordemos que para la suma el orden de los factores no altera el producto, por lo tanto, da igual si sumamos 20 + 10 que 10 +20.

```
print (10 + 20)
```

```
30
```

El operador suma, al igual que los demás operadores que veremos después, se puede emplear para realizar sumas entre dos variables, o variables con números. Por ejemplo, tenemos una variable llamada número de manzanas cuyo valor es cero. Si nosotros queremos añadirle una unidad, basta con sumar 1 a dicha variable. La forma en la que se escribe la sintaxis en Python es similar a con la suma de dos números, basta con escribir el nombre de la variable, seguido del símbolo suma y luego la cantidad a sumar, como se muestra en pantalla.

```
numero_de_manzanas = 0
print (numero_de_manzanas + 1)
```

```
1
```

Si queremos guardar el resultado de la suma directamente en la variable que estamos usando, lo cual muy probablemente será el objetivo en la mayoría de las ocasiones, basta con declarar la variable nuevamente con el resultado de la operación suma, como se puede ver en pantalla.

```
numero_de_manzanas = 0
numero_de_manzanas = numero_de_manzanas + 1
print (numero_de_manzanas)
```

```
1
```

Puede que el ejemplo anterior sea un poco confuso, por lo tanto, iremos por partes. En la primera línea declaramos a la variable número de manzanas como cero, es decir, su valor es cero.

```
numero_de_manzanas = 0
```

Luego, decimos que la variable número de manzanas guarde el valor de la operación suma entre ella misma y 1. Recordemos que, en la línea anterior, número de manzanas vale cero, por lo tanto, es como si le dijéramos a Python que guarde en la variable número de manzanas el valor de 0 + 1.

```
numero_de_manzanas = numero_de_manzanas + 1
                     0   +  1
```

Finalmente, le decimos a Python que nos muestre el valor de número de manzanas, el cual, debido a la operación suma, es 1.

Noten que, como había mencionado anteriormente, el orden de los factores en una suma no afecta el producto, a esa propiedad se le denomina conmutatividad, por lo tanto, la suma es conmutativa. Es por esto por lo que podríamos haber hecho el ejemplo anterior de la forma en la que se muestra en pantalla.

```
numero_de_manzanas = 0
numero_de_manzanas = 1 + numero_de_manzanas
print (numero_de_manzanas)
```

¿Recuerdan que les había dicho que todos los operadores en Python podían operar entre dos variables? La forma en la que se realiza es similar a como ya hemos aprendido. Lo veremos con un ejemplo, queremos sumar el valor de la variable A, que vale 10, (aparece en pantalla A = 10) con el valor de la variable B, que vale 27, (aparece en pantalla B = 27), guardar dicho resultado en una nueva variable llamada respuesta y luego mostrar el valor de la variable respuesta. ¿Crees saber cómo se realiza? Seguramente sí, ¡Trata de hacerlo! (Ponle pausa a este video…). La forma en la que se hace es como aparece en pantalla:

```
a = 10
b = 27
respuesta = a + b
print (respuesta)
```

Quizá hayas pensado en lo siguiente: ¿Y qué pasa si quiero sumar, ya no dos números o variables, sino tres, cuatro o quizá cinco? No hay problema, se sigue la misma sintaxis que ya hemos aprendido, basta con seguir añadiendo sumas en la misma línea, como se muestra en el ejemplo en pantalla:

```
a1 = 10
a2 = 20
a3 = 30
suma = a1 + a2 + a3 + 100
```

```
a1 = 10
a2 = 20
a3 = 30
suma = a1+a2+a3+100
```

1 2 3

Las operaciones de suma se ejecutarán de izquierda a derecha y por pares, es decir, primero será a1 + a2, luego el resultado de esa operación se sumará con a3 y finalmente a dicho resultado se le sumará 100.

5.2.1.1. Sumas entre tipos enteros, flotantes y cadenas

¿Recuerdas que había mencionado que las operaciones aritméticas solo se pueden hacer entre mismos tipos de dato? Esto quiere decir que no es posible, por ejemplo, sumar una cadena de texto con un número entero. No obstante, es posible sumar números flotantes con números enteros, debido a que los números flotantes son una extensión de los enteros. Veamos cómo funciona.

Tenemos la variable x1, que vale 100, y la variable x2 que vale 2.5, la forma en la que se sumaría es similar a como ya hemos aprendido, pero hay una ligera diferencia, el tipo de dato resultante de esa operación será automáticamente modificado a flotante. Podemos comprobar el tipo de dato imprimiendo el resultado de la palabra reservada type.

```
x1 =  100
x2 = 2.5
print (type(x1+x2))
```

```
<class 'float'>
```

Finalmente, y para concluir con el operador suma, con las cadenas de texto o también llamadas strings, el operador suma lo que hace es juntar, pegar ambas cadenas. A este proceso se le llama concatenación.

Por ejemplo, tenemos una variable que almacena una cadena que dice "hola" y otra variable con una cadena que dice "adiós". Si sumamos ambas variables obtendremos lo siguiente:

```
v1 = "hola"
v2 = "adiós"
print (v1 + v2)
```

```
holaadiós
```

5.2.2. Resta

Similar a la suma, y como será con los demás operadores aritméticos, la sintaxis, es decir, la forma de escribir la operación es idéntica, lo único que cambia es el símbolo de la operación, en este caso es el símbolo menos (-), por lo tanto, se detallarán los aspectos propios del operador resta. Sin más preámbulo, comenzamos.

Lo primero y más importante a tener en cuenta es que la resta, como lo es en las matemáticas, NO es una operación conmutativa. ¿Recuerdas lo que era la conmutatividad? Si no lo recuerdas, no te preocupes que yo te recuerdo. La conmutatividad es la característica de que el orden de los factores no altera el producto. La resta, como había mencionado, no es conmutativa, es decir, el orden en el que se escriba la operación SÍ influye e importa. Si no has entendido no te preocupes, te mostraré un ejemplo:

Tenemos la variable A igual a 5 y la variable B igual a 10. Si restamos A-B nos dará -5.

```
a = 5
b = 10
print (a - b)
```

```
-5
```

Por otra parte, si restamos B-A, el resultado será 5.

```
a = 5
b = 10
print (b - a)
```

```
a = 5
b = 10
print (b - a)
```

```
5
```

No es lo mismo, ¿verdad? Es por esto por lo que con el operador resta hay que tener cuidado del orden.

Con la resta ocurre algo similar a la suma al momento de operar entre un tipo entero y un flotante, el resultado será un número flotante, como se ve en el ejemplo en pantalla.

```
a = 2.5
d = 5
print (type(b-a)
```

```
<class 'float'>
```

Aplicar la operación de resta a variables es igual que como se haría con una suma, basta con poner el símbolo resta (-) en lugar al de suma. Es importante que quede bien claro este concepto, ya que de la misma forma funcionan todos los demás operadores, no solo los aritméticos, también los lógicos y de comparación.

```
a = 2.5
b = 5
c = b - a
print (c)
```

```
2.5
```

Para terminar con el operador resta, por si tenías curiosidad, la resta **NO** funciona entre cadenas de texto.

5.2.3. Multiplicación

La multiplicación matemáticamente se define como la suma de un número tantas veces como lo indique el otro número, hace parte de los operadores aritméticos que incluye Python y es muy sencillo de usar.
Para utilizar la multiplicación basta con usar la sintaxis que ya hemos aprendido para hacer operaciones, usando el símbolo asterisco (*), como se muestra en el ejemplo en pantalla:

```
variable1 = 2
variable2 = 3
print (variable1 * variable2)
```

```
6
```

Al igual que el operador suma, este operador es conmutativo, es decir, el orden de los factores no altera el producto. ¿Sí recordabas lo que significaba conmutativo? Como te estarás dando cuenta, es un concepto importante.

Como la multiplicación es conmutativa, no importa el orden en el que hagamos la operación, para un ejemplo demostrativo, podemos tomar el ejemplo anterior e invertir el orden de las variables a la hora de multiplicar:

```
variable1 = 2
variable2 = 3
print (variable2 * variable1)
```

```
variable1 = 2
variable2 = 3
print (variable2 * variable1)
```

```
6
```

El comportamiento de la operación entre un tipo entero y uno flotante nos dará un resultado flotante, como era de esperar.

5.2.4. División

El operador división similar a la resta, NO es conmutativo. El orden de la operación influye en el resultado.

La división sigue la sintaxis que ya hemos aprendido, utilizando el símbolo slash (/). La forma de emplearse es como la que se muestra en pantalla:

```
print (11 / 2)
```

```
5.5
```

La división tiene una característica muy importante a resaltar: esta operación siempre resultará en un valor flotante, independientemente de si los números o variables operadas son ambas enteras o cuyo resultado es entero. Por ejemplo, si dividimos 10 entre 5 el resultado es 2, sin embargo, el 2 será de tipo flotante en lugar de tipo entero. Dicha característica hay que tenerla en cuenta en ciertas situaciones.

Para comprobar lo anteriormente mencionado, haremos el siguiente ejemplo, declararemos dos variables, A y B, estas variables tendrán como valor 100 y 20, respectivamente. Luego, mediante el comando type, verificaremos el tipo de ambas variables. Por último, realizaremos la división de A dividido por B y veremos el tipo de dato del resultado. (Si es un video, los pasos irían mostrándose en pantalla conforme voy hablando)

```
A = 100
B = 20
print (type (A))
print (type (B))
print (type (A/B))
```

```
<class 'int'>
<class 'int'>
<class 'float'>
```

Finalizando con el operador división, es muy importante saber que no es posible dividir entre cero, esto es debido a que, matemáticamente, la división por cero no está definida. Si tratas de dividir un número entre cero, ya sea por accidente o con intención, te saldrá un error llamado ZeroDivisionError.

5.2.5. División Entera

¿Recuerdas que la división en Python siempre retorna un resultado flotante? Pues, el operador división entera funciona similar a la división con un cambio importante: el resultado que este retorna siempre será entero. Esto tiene algunas implicaciones que hay que aclarar que veremos en ejemplos por separado.

Lo primero que se debe saber es cómo se aplica el operador, este sigue la sintaxis que ya hemos aprendido, usando dos veces el símbolo de la división juntos, como se muestra en el ejemplo en pantalla:

```
print (10 // 2)
```

```
5
```

La primera implicación es que, para divisiones cuyo resultado sea flotante, siempre será redondeado hacia abajo, es decir, supongamos que nuestro resultado nos da 1.98. Si ese resultado fue empleado por una división entera, lo que nos devolverá será 1. Hagamos otro ejemplo, imprimamos el resultado de 2 dividido entre 3, como se ve en pantalla:

```
print (2/3)
```

```
0.6666666666666666
```

La primera implicación es que, para divisiones cuyo resultado sea flotante, siempre será redondeado hacia abajo, es decir, supongamos que nuestro resultado nos da 1.98. Si ese resultado fue empleado por una división entera, lo que nos devolverá será 1. Hagamos otro ejemplo, imprimamos el resultado de 2 dividido entre 3, como se ve en pantalla:

Vemos que el resultado es 0.66 periódico. Ahora, si repetimos la misma prueba, pero en este caso usando el operador de división entera, veremos que el resultado obtenido es cero, como se muestra en pantalla. Esto es debido a que la parte entera de la fracción es cero y, como había mencionado anteriormente, Python siempre redondeará con ese operador hacia abajo.

```
print (2//3)
```

```
0
```

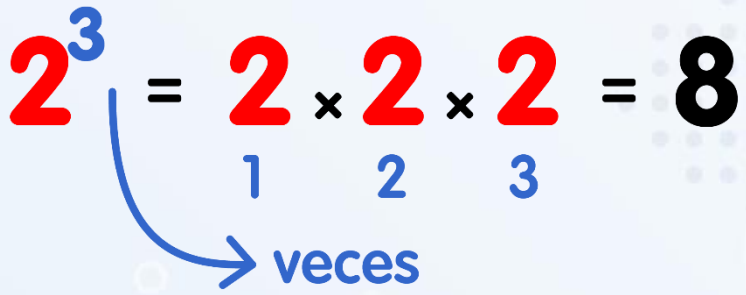
Para finalizar respecto al operador de división entera, al igual que con una división normal, no es posible dividir entre cero, debido a que la división entre cero es matemáticamente indefinida.

5.2.6. Potenciación

Quizá no recuerdes en lo que consiste hacer una potenciación, el concepto es sencillo, te lo voy a recordar. Hay que tener en cuenta dos factores, la base, representado de color rojo, y el exponente, representado con el color azul. Matemáticamente se escribe como se muestra en pantalla.

$$b^e$$

La función de la potenciación es multiplicar la base, por sí mismo, las veces que indique el exponente, como se muestra en pantalla:



La potenciación no es conmutativa, por lo tanto, el orden importa. En Python, la operación de potenciación usa la misma sintaxis que ya hemos aprendido, usando dos veces el símbolo de multiplicar juntos, donde la base estará al lado izquierdo del operador y el exponente al lado derecho. Insisto, la potenciación no es conmutativa, por lo tanto, el orden importa.

```
print (2**3)
```

```
8
```

```
print (3**2)
```

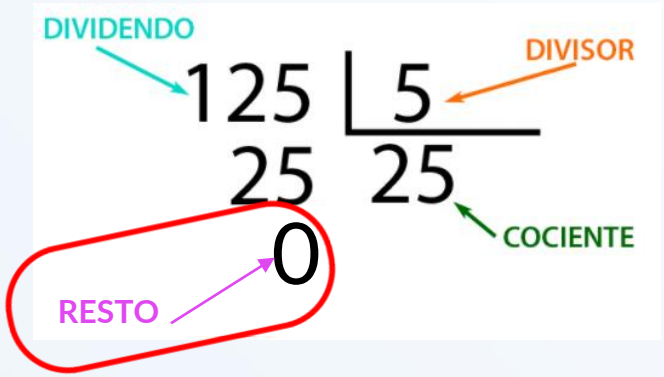
```
9
```

En Python, todo número cuyo exponente sea cero es igual a uno, incluyendo el cero, es decir, Python devuelve cero elevado a la cero igual a 1.

5.2.7. Módulo

El módulo es el operador más desafiante de entender, pero es uno muy útil en determinadas situaciones, tanto es así que hay una rama de las matemáticas dedicada a ello. Si te da curiosidad, esta rama se llama teoría de números.

El operador módulo tiene como función devolver el residuo, o también llamado resto, de una división. Quizá no recuerdes qué es eso, pero no te preocupes, en pantalla te muestro lo que es.



El operador módulo no es conmutativo, el orden importa. En Python, este operador se escribe con la sintaxis que ya hemos aprendido, usando el símbolo porcentaje (%)

```
print (125%5)
```

```
0
```

El módulo tiene muchas utilidades, una de ellas, de las más comunes, es determinar si un número es múltiplo de otro. Por ejemplo, si queremos saber si un número es par, es decir, que sea múltiplo de dos, podemos verificar que el módulo entre dos sea cero. Ejemplos prácticos de esto lo veremos después de ver los operadores de comparación.

5.3. Operadores de comparación

Los operadores de comparación, como su nombre lo indica, sirven para comparar entre, al menos, un par de valores. En la vida real, nosotros muchas veces tomamos decisiones basadas en comparaciones, por ejemplo, cuando queremos decidir en si comprar un mismo producto entre una tienda y otra, generalmente decidiremos por la que tenga un menor precio.

Los operadores de comparación siempre nos van a devolver un valor booleano, si se cumple la comparación nos devolverá verdadero, True, o, en caso contrario, falso, False.

Todos los operadores de comparación siguen la misma sintaxis de los operadores aritméticos, es decir, se escribe un valor, número o variable a la derecha, luego el símbolo del operador, y por último el otro valor, número o variable.

Veamos cuáles operadores de comparación nos ofrece Python.

5.3.1. Igualdad

Empecemos por el operador más sencillo, el de igualdad. Esto es bastante sencillo, lo que hace es comparar si ambos valores dados para el operador son iguales, en caso de serlo, devolverá True, en caso contrario, devolverá False. Veamos un ejemplo.

Tenemos dos variables, X y Y, ambas valen 5. Queremos saber si son iguales. La forma en la que en Python se realiza esta operación es usando el operador de igualdad, cuyo símbolo es dos iguales juntos (==). Entonces, quedaría como se muestra en pantalla.

```
X = 5
Y = 5
print (X==Y)
```

```
True
```

Ahora, cambiemos el valor de Y a 6, veremos que el resultado ahora será falso porque cinco no es igual a seis.

```
X = 5
Y = 6
print (X==Y)
```

```
False
```

Sencillo, ¿no? Continuemos con el siguiente operador.

5.3.2. Diferencia

El operador de diferencia es igual de sencillo que el de igualdad, es más, hace lo mismo, pero… al contrario; es decir, en lugar de verificar si son iguales, verifica que son diferentes. En últimas, lo opuesto al operador de igualdad.

El símbolo de este operador es el de cerrar exclamación con un igual juntos (!=), como se muestra en pantalla.

Tomemos el mismo ejemplo anterior para verlo mejor y en acción.

Tenemos dos variables, X y Y, con el valor de 5 y 6, respectivamente. Si imprimimos X es diferente a Y, nos devolverá True, porque efectivamente cinco es diferente a seis.

```
X = 5
Y = 6
print (X!=Y)
```

```
True
```

Evidentemente si ponemos algo, por ejemplo, "¿es cinco diferente de cinco?" nos devolverá falso. Igual de sencillo que el operador de igualdad, ¿no? Continuemos.

5.3.3. Mayor que

El siguiente operador de comparación es el de mayor que. Este operador nos devuelve True si el número de la izquierda es mayor que el de la derecha, False en caso contrario. Su símbolo es idéntico al mayor que de las matemáticas (>).

Su uso es similar al de los demás operadores, veamos un ejemplo.

En pantalla se muestra cómo le preguntaríamos a Python si 20 es mayor que 10 e imprima el resultado.

```
print (20 > 10)
```

```
True
```

El siguiente operador es el opuesto a este.

5.3.4. Menor que

El operador menor que es el opuesto al mayor que, es decir, devolverá True si el número de la izquierda es menor al de la derecha, False en caso contrario. Similar al operador anterior, su símbolo es idéntico al menor que el de las matemáticas (<).

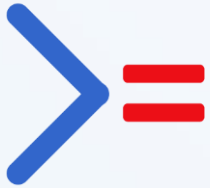
En pantalla se muestra cómo le preguntaríamos a Python si la variable X, que vale 5, es menor que la variable Y, que vale 1 y que nos imprima en pantalla el resultado. Vemos que la respuesta es False, porque efectivamente cinco NO es menor que uno.

```
X = 5
Y = 1
print (X<Y)
```

```
False
```

5.3.5. Mayor o igual que

Este operador es similar al mayor que, la única diferencia es que también devolverá True si ambos números son iguales. Su símbolo es el mayor que junto con un igual. (>=)



Un ejemplo de su diferencia con el mayor que, es el siguiente: Tenemos dos variables, X y Y, ambas tienen el valor de 5. Queremos ver si X es mayor o igual que Y e imprimir el resultado de la operación.

```
X = 5
Y = 5
print (X>=Y)
```

```
True
```

Vemos que el resultado es verdadero, True, esto es debido a que, aunque cinco no sea mayor que cinco, sí son iguales. Nótese que el operador se llama mayor O igual. Es por esto por lo que, si se cumple alguna de estas dos condiciones, el resultado es verdadero.

5.3.6. Menor o igual que

Al igual que pasa con el operador anterior, el menor o igual que, es similar al menor que, con la diferencia de que devolverá True si ambos números son iguales. Su símbolo es el menor que junto con igual. (<=)

Veamos su funcionamiento, tenemos una variable llamada B cuyo valor es 10, queremos imprimir si B es menor o igual que 50. En pantalla se muestra el código.

```
B = 10
print (B<=50)
```

```
True
```

El resultado es True, verdadero, esto es debido a que, aunque 10 no sea igual a 50, sí es menor que 50. Recordemos que el operador es menor O igual que, por lo tanto, al cumplirse al menos una de las dos condiciones, el resultado es verdadero.

Este era el último operador de comparación, son sencillos, ¿no? Continuaremos con los operadores lógicos.

5.3.7. Resumen

Como resumen de los operadores de comparación, podemos decir lo siguiente: Todos los operadores de comparación evalúan dos datos y devolverán un dato de tipo Booleano (verdadero o falso) dependiendo de si se cumple el operador o no. A continuación, te muestro una pequeña tabla bastante útil:

Operador	Nombre	Ejemplo	Equivalencia natural
==	Igualdad	x == y	¿Es X igual a Y?
!=	Diferencia	x != y	¿Es X diferente a Y?
>	Mayor que	x > y	¿Es X mayor que Y?
<	Menor que	x < y	¿Es X menor que Y?
>=	Mayor o igual	x >= y	¿Es X mayor O igual que Y?
<=	Menor o igual	x <= y	¿Es X menor O igual que Y?

Operador	Nombre	Ejemplo	Equivalencia natural
==	Igualdad	x == y	¿Es X igual a Y?
!=	Diferencia	x != y	¿Es X diferente a Y?
>	Mayor que	x > y	¿Es X mayor que Y?
<	Menor que	x < y	¿Es X menor que Y?
>=	Mayor o igual	x >= y	¿Es X mayor O igual que Y?
<=	Menor o igual	x <= y	¿Es X menor O igual que Y?

5.4. Operadores lógicos

Los operadores lógicos son muy especiales, dado que por sí solos no tienen mucha utilidad; pero, al usarse junto con operadores de comparación se logra una sinergia increíblemente útil. Python incluye tres operadores lógicos base: el AND, OR y NOT; Estos tres operadores lógicos son los más básicos de todos los operadores lógicos, debido a que con ellos uno puede crear otros operadores lógicos de mayor complejidad. No obstante, nosotros estamos interesados sólo en esos tres ya que son los más importantes.

Como iba diciendo, los operadores lógicos tienen una sinergia con los operadores de comparación, esto es debido a que se complementan el uno a otro para hacer operaciones más interesantes mediante la concatenación de operadores de comparación. ¿Recuerdas la palabra concatenar? En caso de que no, consiste en unir dos o más partes, en este caso, los operadores lógicos sirven para unir operadores de comparación.

Ten muy en cuenta lo siguiente: **Los operadores lógicos funcionan solo con valores o variables booleanas, a su vez, devuelven como respuesta valores booleanos.**

Todo esto puede sonar muy críptico o confuso, pero no te preocupes, en realidad es muy sencillo, así que, sin más preámbulo, comencemos.

5.4.1. Operador AND

El operador AND, que al español significa "y", es un operador muy especial, este operador solo devolverá True si ambos valores, el de la izquierda y el de la derecha, son verdaderos. Para usar este operador basta con escribir un valor booleano, luego la palabra reservada "and" en minúscula y por último otro valor booleano, como se muestra en el ejemplo en pantalla:

```
print (True and False)
```

```
False
```

Como puedes ver, devuelve False debido a que el operador AND solo devolverá verdadero si ambos valores son verdaderos. Esto nos permite hacer una pequeña tabla que nos resuma su comportamiento. A esta tabla la llamamos "tabla de la verdad", y como es específicamente para el operador AND, es la tabla de la verdad del AND.

Izquierda	Derecha	Respuesta
True	True	True
True	False	False
False	True	False
False	False	False

q	p	r
True	True	True
True	False	False
False	True	False
False	False	False

Teniendo en cuenta la tabla de verdad del AND, podemos deducir que si escribimos en Python “falso y verdadero” nos devolverá falso, como se puede ver en pantalla:

```
print (False and True)
```

```
False
```

```
print (False and True)
```

```
False
```

O, por el contrario, si escribimos “verdadero y verdadero”, nos devolverá verdadero.

```
print (True and True)
```

```
True
```

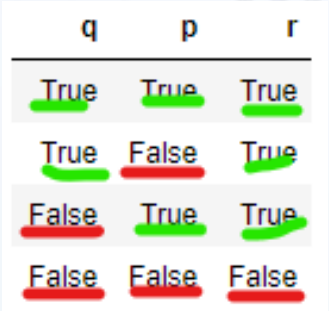
```
print (True and True)
```

```
True
```

5.4.2. Operador OR

El operador OR, que en español significa "o", es un operador que, al contrario del AND, devuelve a todo verdadero a menos que ambos valores, el de la izquierda y la derecha del operador, sean falsos. Veamos la tabla de la verdad del OR para que se entienda rápidamente:

Izquierda	Derecha	Respuesta
True	True	True
True	False	True
False	True	True
False	False	False



q	p	r
True	True	True
True	False	True
False	True	True
False	False	False

Interesante, ¿verdad? Es por esto por lo que el operador de comparación mayor o igual y menor o igual devuelven verdadero si alguna de las dos condiciones se cumple. Noten que incluso usan el operador lógico OR en el mismo nombre, "Mayor **O** igual", "Menor **O** igual".

Es más, ahora que conocemos las tablas de verdad del operador AND y OR, podemos empezar a ver esa poderosa sinergia entre los operadores de comparación con los operadores lógicos.

Haremos como ejemplo lo que haría un operador "Mayor o igual que" sin usar directamente ese operador. Sabemos que los operadores de comparación devuelven valores booleanos, justo lo que necesitan nuestros operadores lógicos para funcionar, entonces podemos ponernos manos a la obra.

Tenemos dos variables, X y Y, ambas con el valor de cinco. Vamos a imprimir en pantalla si X es mayor o igual que Y, si usar el operador de comparación "mayor o igual". Primero, tendríamos que hacer la comparación de si X es mayor que Y, luego, concatenar dicha comparación con el operador OR junto con la comparación de si X es igual a Y, como se muestra en pantalla.

```
X = 5
Y = 5
print (X > 5 or X == 5)
```

```
True
```

```
X = 5
Y = 5
print (X > 5 or X == 5)
```

```
True
```

Vemos que el resultado de la operación lógica es True. Esto es porque, a pesar de que cinco no sea mayor que cinco, sí se cumple el operador lógico de la derecha, ya que cinco sí es igual a cinco. Todo gracias a que estamos usando el operador OR. Si lo hubiésemos hecho con el operador AND, nos hubiese devuelto False, ya que, recuerden, el operador AND solo devuelve verdadero si ambos valores son verdaderos.

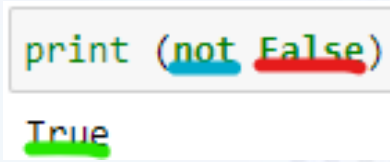
5.4.3. Operador NOT

El tercer operador lógico es el NOT, que en español significa “no”. Este operador es muy especial, dado que por sí solo no sirve para hacer ninguna operación, lo único que hace es negar, o más específicamente, invertir el valor booleano que viene después de él. Esto puede sonar un poco confuso, así que aquí les va un ejemplo.

Queremos que Python imprima en pantalla el inverso de False, es decir, que nos imprima True, sin imprimir la palabra reservada True. El código sería el siguiente:

```
print (not False)
```

```
True
```



Vemos que estamos negando el False, lo cual inmediatamente lo convierte en un True. Este peculiar comportamiento de invertir es muy útil en ciertos casos, donde trabajar con el valor original sea más difícil.

En el tema que viene a continuación, la precedencia de operadores, veremos combinaciones de todo lo que hemos aprendido durante esta sección, ¡Qué emoción!

5.5. Precedencia de operadores

5.5.1. Precedencia matemática PEMDAS

En las matemáticas, el orden en el que se realizan las operaciones importa muchísimo, debido a que, por poner un ejemplo, una mala interpretación por parte del usuario en una fórmula matemática puede llevar a un comportamiento totalmente indeseado e inesperado. Esta clase de errores se dan más comúnmente de lo que uno esperaría, en parte porque mucha gente hace caso omiso o directamente no recuerda las reglas que hay que seguir cuando se enfrentan a expresiones matemáticas.

En esta pequeña sección veremos por qué es muy importante tener claro el orden en el que se deben realizar las operaciones. Para ello, empezaremos con un ejemplo que a su vez es una pequeña prueba.

¿Cuál es el resultado de esta operación?

7-2(13+5)

Si tu respuesta es 90 estás equivocado. La respuesta correcta es -29. ¿Por qué? Por el orden en el que hay que realizar las operaciones. Veámoslo con detalle.

El orden con el que se deben de realizar las operaciones, de izquierda a derecha, es el siguiente:

1.1. Paréntesis

2. Exponentes (o raíces)

3. Multiplicación y división

1.4. Adición y sustracción (sumas y restas)

Ese orden se puede conocer de forma muy sencilla acordándose del siguiente acrónimo: PEMDAS. Donde cada letra del acrónimo hace referencia al orden de las operaciones, en el orden respectivo.

Ahora, teniendo en cuenta a PEMDAS, resolvamos nuevamente el ejemplo anterior.

Primero debemos resolver los paréntesis, la expresión quedaría así:

$$7 - 2(18)$$

Luego vienen los exponentes, que en este caso no tenemos ninguno, por lo tanto, podemos proceder con las multiplicaciones y divisiones. ¿Ves que el 2 está siendo multiplicado por 18? El resultado de esa multiplicación es 36. Nuestra expresión en este momento iría así:

$$7 - 36$$

Por último, resolvemos las sumas y las restas. En este caso es una resta, recuerda que las restas no son conmutativas, por lo tanto, el orden importa. No es lo mismo 7-36 que 36-7.

Ahora que conoces al PEMDAS, ponte a prueba con la siguiente expresión:

$$50-(3(2_3+1)-7)$$

La respuesta correcta es 30. ¿Lo lograste?

5.5.2. Precedencia aplicada en Python

Como vimos anteriormente, el orden de realización de las operaciones importa, por lo tanto y como era de esperar, Python sigue muy estrictamente dicho orden, con un pequeño agregado: podemos usar los paréntesis en Python sin necesidad de usar operaciones matemáticas para elevar el privilegio de operación. Esto significa que, incluso al no hacer operaciones matemáticas, podemos hacer que Python ejecute otros operadores de primero, con respecto a otros, si encerramos dicha operación dentro de un paréntesis. Esto lo veremos más detalladamente con unos ejemplos de operadores lógicos.

¿Qué crees que se imprimirá en pantalla con el siguiente código? Trata de realizar la operación en tu cabeza.

```
X = True
Y = False
Z = False
print (not Z and (X or Y))
```

La respuesta es True. Fíjate en el paréntesis del "X or Y", esto hace que se realice esta operación primero que las demás. Como X es verdadero y Y es falso, el operador OR, como ya hemos aprendido, devolverá verdadero, luego, las demás operaciones tienen el mismo nivel de prioridad, por lo tanto, se resolverán de izquierda a derecha. El operador NOT invierte el valor booleano de Z, convirtiéndolo en True, luego se evalúa el operador AND, donde a la izquierda tenemos True y a la derecha, por el resultado del paréntesis, también. Verdadero y verdadero es verdadero, recuerda la tabla de verdad del AND.

¿Qué crees que imprimirá en pantalla el siguiente código?

```
X = True
Y = False
Z = False
print (not(not Z and not(X or Y)))
```

La respuesta es True. Trata de realizar la operación en tu mente. ¡Ten cuidado con los paréntesis!

Ahora que sabes aplicar los paréntesis a operaciones en Python, veamos cómo eran los dos ejemplos de la sección anterior escritos en código Python.

1. 7-2(13+5)

```
print (7-2*(13+5))
```

```
-29
```

2. 50-(323+1-7)

```
print (50-(3*(2**3+1)-7))
```

```
30
```

Material complementario

Curso Python desde cero #6 | Operadores aritméticos en Python - YouTube

8. Programación en Python | Operadores Lógicos – YouTube

Referencias Bibliográfica

3. An Informal Introduction to Python — Python 3.9.4 documentation

Built-in Types — Python 3.9.4 documentation