Esta clase va a ser

- grabada

Clase 03. PYTHON

Operadores básicos y expresiones anidadas

CODERHOUSE

Temario

02

Listas y Tuplas

- ✓ Listas
- ✓ Funciones
- ✓ Tuplas

03

Operadores y expresiones

- ✓ Operadores
- ✓ Expresiones anidadas

04

Controladores de Flujo I

- ✓ Condicional
- ✓ Else
- ✓ Elif

Objetivos de la clase

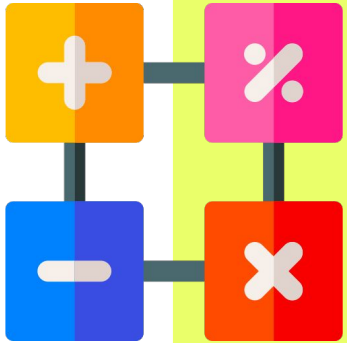
- **Reconocer** un Operador
- **Identificar** similitudes y diferencias entre operador y expresión
- **Reconocer** expresiones.

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:

Aritméticas	Algebraicas
si ambos operandos son valores literales:	si al menos un operando es una variable:
2 + 5 -1.4 * 54 1/2.5	radio * 3.14 (nota_1 + nota_2)/2

El tipo lógico

CODERHOUSE

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.

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.

Tipo lógico

Lingüístico	Matemático
En contexto lingüístico podríamos decir que:	Y en contexto matemático:
“Estoy vivo” es Verdadero (True)	1 +1 = 3??? es Falso (False)

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

CODERHOUSE

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**).

CODERHOUSE

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 (==).

Ejemplo:

```
>>> a = 3
>>> a == 3
True
```

No confundir el operador de asignación (=) con el operador de igualdad (==)

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 sólo 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 Relacionales

Calcular el resultado de cada expresión

Duración: **10 minutos.**

ACTIVIDAD EN CLASE

Operadores relacionales

En una lista encontraremos diferentes operaciones relacionales, calcular mentalmente el resultado de cada expresión y almacenarlo en una nueva lista que contendrá únicamente valores lógicos True y False.

ACTIVIDAD EN CLASE

Operadores relacionales

Sugerencia. Si necesitas ayuda, deja que Python calcule estas expresiones por ti

```
expresiones = [
        False == True,
        10 >= 2*4,
        33/3 == 11,
        True > False,
        True*5 == 2.5*2
]
```

Operadores lógicos

CODERHOUSE

Operadores Lógicos

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

Not

(no - negación)

Or

(o de esto o aquello)

And

(Y de esto y eso).

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)
- Sólo afecta a los lógicos

Más operadores

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

Conjunción	Disyunción
Viene de conjunto	Viene de disyunto
Sinónimo de unido, contiguo	Sinónimo de separado
Agrupa uniendo	Agrupa separando

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

And

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 RECORDAR

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

Expresión	Resultado
True and True	True
True and False	False
False and True	False
False and False	False

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

p	q	p ∧ q
V	V	
V	F	
F	V	
F	F	

p	q	p ∨ q
V	V	
V	F	
F	V	
F	F	

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 RECORDAR

Para Python, una separación **or** es solamente verdadera **(True)** cuando, y solo 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

Expresión	Resultado
True o True	True
True o False	True
False o True	True
False o False	False

Tabla de verdad del Or

Su tabla de verdad quedaría así:

p	q	p ∨ q
V	V	
V	F	
F	V	
F	F	

Operadores Lógicos

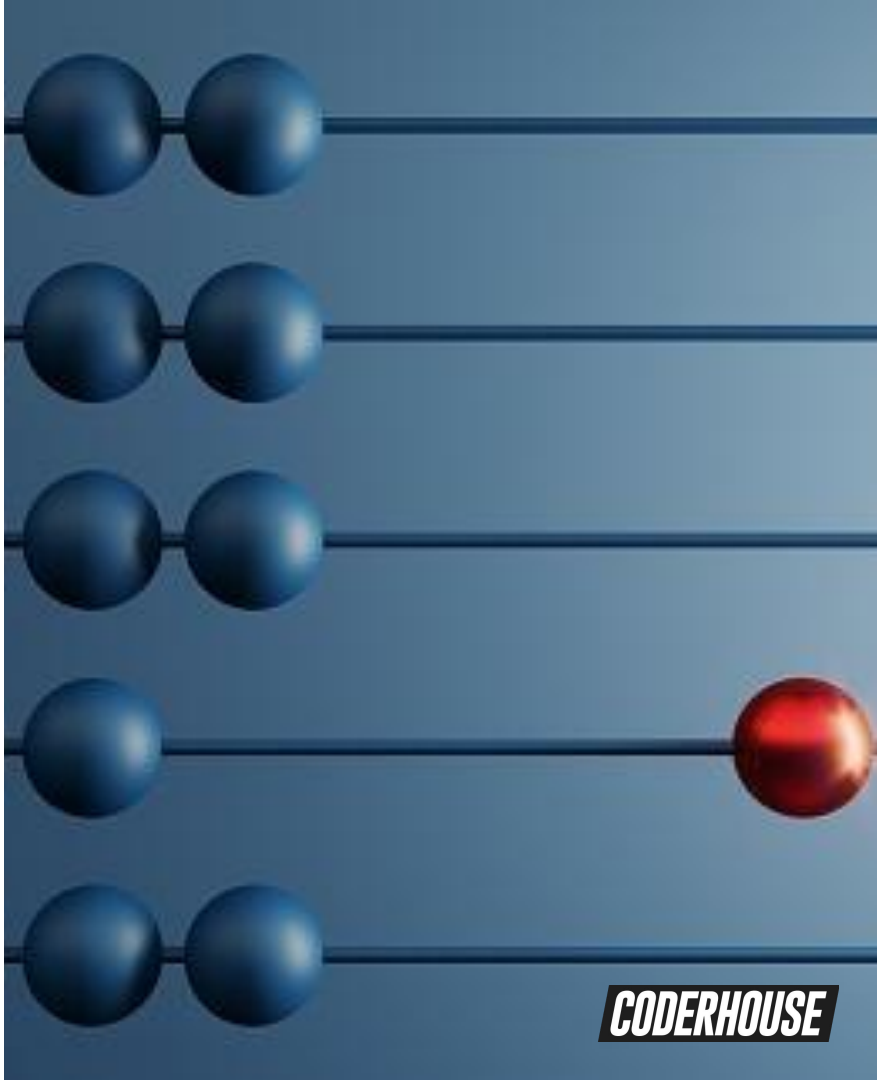
Calcular el resultado de cada expresión y almacenarlo en una nueva lista

Duración: **10 minutos**

ACTIVIDAD EN CLASE

Operadores Lógicos

1. En una lista encontraremos diferentes operaciones lógicas. Calcular mentalmente el resultado de cada expresión y almacenarlo en una nueva lista la cual contendrá únicamente valores lógicos **True** y **False**.

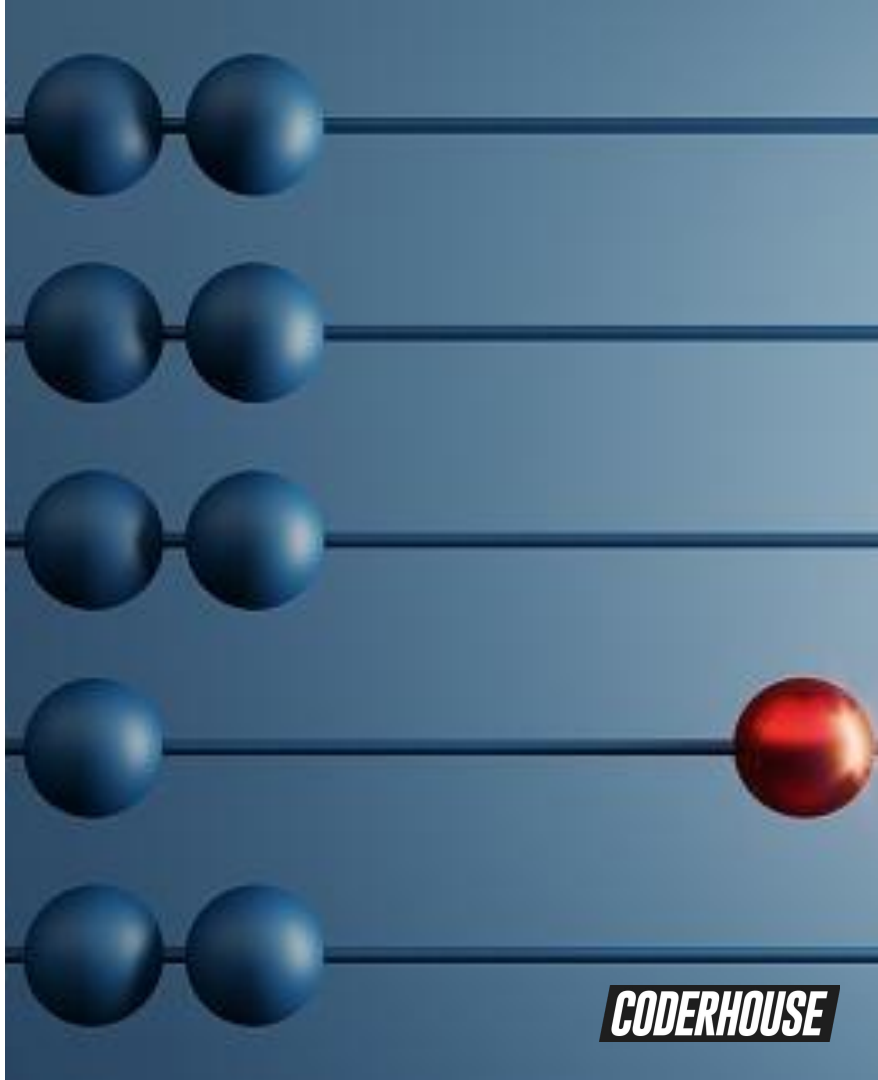


ACTIVIDAD EN CLASE

Operadores Lógicos

Sugerencia: si necesitas ayuda, deja que Python calcule estas expresiones por ti!

```
expresiones = [
  i.   not False,
  ii.  not 3 == 5,
  iii. 33/3 == 11 and 5 > 2,
  iv.  True or False,
  v.   True*5 == 2.5*2 or 123 >= 23,
  vi.  12 > 7 and True < False
]
```



Break

¡10 minutos y volvemos!

Expresiones anidadas

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).

Expresiones anidadas

Crear una variable que almacene si se cumplen **todas** las condiciones

Duración: **10 minutos**

ACTIVIDAD EN CLASE

Expresiones anidadas

A partir de dos variables llamadas **NOMBRE** y **EDAD**, debes crear una variable que almacene si se cumplen todas las siguientes condiciones, encadenando operadores lógicos en una sola línea:

- ✔ **NOMBRE** sea diferente de cuatro asteriscos “****”
- ✔ **EDAD** sea mayor que 5 y a su vez menor que 20
- ✔ Que la **longitud** de NOMBRE sea mayor o igual a 4 pero a la vez menor que 8
- ✔ **EDAD** multiplicada por 3 sea mayor que 35

Desde un input conseguir las variables:
nombre = INPUT!!!
edad = INPUT!!!!

CODERHOUSE

Operadores de asignación

CODERHOUSE

Operadores de asignación

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.

CODERHOUSE

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

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

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

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

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

Operador	Ejemplo	Equivalente
=	a = 2	a = 2
+=	a += 2	a = a + 2
-=	a -= 2	a = a −2
*=	a *= 2	a = a * 2
/=	a /= 2	a = a / 2
%=	a %= 2	a = a % 2
**=	a **= 2	a = a ** 2

¿Quieres saber más?
Te dejamos material ampliado de la clase

CODERHOUSE

MATERIAL AMPLIADO

Recursos

- ✓ Operadores relacionales
- ✓ Tipo Lógico
- ✓ Operadores Lógicos
- ✓ Operadores Python
- ✓ Tipo Lógico
- ✓ Operadores de asignación

Disponible en nuestro repositorio.

¿Aún quieres conocer más? Te recomendamos el siguiente material

CODERHOUSE

MATERIAL AMPLIADO

Recursos multimedia

- ✓ Variables Lógicas | **Nicolás Perez**
- ✓ EjemploClase

Disponible en nuestro repositorio.

CODERHOUSE

¿Preguntas?

Resumen
de la clase hoy

- ✓ Operador Relacional
- ✓ Tipo lógico
- ✓ Operador Lógico
- ✓ Expresiones Anidadas
- ✓ Operador en asignación

Opina y valora esta clase

CODERHOUSE

Muchas gracias.

#DemocratizandoLaEducación