Booleano

El tipo `bool` es un tipo primitivo fundamental en Rust que representa valores lógicos. Este tipo es esencial para expresar condiciones, realizar comparaciones y controlar el flujo de ejecución de un programa mediante estructuras condicionales y bucles.

Un valor booleano solo puede adoptar dos estados posibles:

- `true` : verdadero
- `false` : falso

```
let es_mayor_edad: bool = true;
let esta_lloviendo: bool = false;
let resultado: bool = 10 > 5;  // true
```

El tipo `bool` ocupa exactamente 1 byte de memoria, aunque conceptualmente solo requiere 1 bit de información.

Operadores Lógicos

Rust proporciona un conjunto de operadores lógicos para combinar, invertir o comparar valores booleanos. Estos operadores son fundamentales para construir expresiones lógicas complejas.

Operador	Nombre	Descripción
!	NOT : Negación	Invierte el valor lógico: `true` se convierte en `false` y viceversa
&&	AND : Conjunción	Devuelve `true` solo si ambos operandos son `true`
||	OR : Disyunción	Devuelve `true` si al menos uno de los operandos es `true`
^	XOR : Disyunción exclusiva	Devuelve `true` solo si exactamente uno de los operandos es `true`

Tabla 1: Operadores lógicos para el tipo `bool`

NOT

El operador `!` invierte el valor de una expresión booleana. Es un operador unario que se aplica a un único operando.

Expresión	Resultado
`!true`	`false`
`!false`	`true`

Tabla 2: Tabla de verdad del operador NOT

AND

El operador && evalúa dos expresiones y devuelve `true` únicamente cuando ambas expresiones son verdaderas. Este operador implementa **short-circuit**: si el primer operando es `false`, el segundo no se evalúa.

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

Tabla 3: Tabla de verdad del operador AND

Ejemplo práctico:

```
let edad = 25;
let tiene_licencia = true;
let puede_conducir = edad >= 18 && tiene_licencia;
println!("¿Puede conducir? {}", puede_conducir);  // true
```

OR

El operador `||` devuelve `true` si al menos uno de los operandos es verdadero. También implementa evaluación perezosa: si el primer operando es `true`, el segundo no se evalúa.

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

Tabla 4: Tabla de verdad del operador OR

Ejemplo práctico:

```
let es_fin_semana = true;
let es_feriado = false;
let puede_descansar = es_fin_semana || es_feriado;
println!("¿Puede descansar? {}", puede_descansar);  // true
```

XOR

El operador ^ devuelve `true` únicamente cuando los operandos tienen valores diferentes. Es útil para detectar discrepancias o alternar estados.

A	B	A ^ B
true	true	false
true	false	true
false	true	true
false	false	false

Tabla 5: Tabla de verdad del operador XOR

Ejemplo práctico:

```
let estado_anterior = true;
let estado_actual = false;
let hubo_cambio = estado_anterior ^ estado_actual;
println!("¿Hubo cambio? {}", hubo_cambio);  // true
```

Operadores de Comparación

Los operadores de comparación evalúan relaciones entre valores y devuelven un resultado booleano. Son fundamentales para la construcción de expresiones condicionales.

Operador	Nombre	Descripción
==	Igualdad	Verdadero si ambos valores son iguales
!=	Desigualdad	Verdadero si los valores son diferentes
<	Menor que	Verdadero si el izquierdo es menor que el derecho
>	Mayor que	Verdadero si el izquierdo es mayor que el derecho
<=	Menor o igual	Verdadero si el izquierdo es menor o igual al derecho
>=	Mayor o igual	Verdadero si el izquierdo es mayor o igual al derecho

Tabla 6: Operadores de comparación

Conversiones y Métodos

Conversión a Cadena

El tipo `bool` puede convertir valores booleanos a cadenas de texto:

```
let verdadero = true;
let falso = false;

println!("{}", verdadero.to_string());  // "true"
println!("{}", falso.to_string());      // "false"
```

Conversión a Enteros

Los valores booleanos pueden convertirse a tipos numéricos mediante el operador `as`. Por convención, `false` se convierte en `0` y `true` en `1`:

```rust
let verdadero = true;
let falso = false;

let valor_v: u8 = verdadero as u8;    // 1
let valor_f: u8 = falso as u8;        // 0
```

Precedencia de Operadores

Cuando se combinan múltiples operadores, es importante comprender su orden de evaluación:

1. `!`: Mayor precedencia
2. Operadores de comparación (`==`, `!=`, `<`, `>`, `<=`, `>=`)
3. `&&`
4. `||`
5. `^`: Menor precedencia

```rust
let resultado1 = !false && true || false;
// Equivale a: ((!false) && true) || false
// = (true && true) || false
// = true || false
// = true

// Con paréntesis explícitos
let resultado2 = (!false && true) || false;
// true
```

Recomendación: Aunque Rust tiene reglas claras de precedencia, el uso de paréntesis explícitos mejora la legibilidad del código y previene errores sutiles.