Punto Flotante

Los tipos de punto flotante (**floating-point**) permiten representar números con componente fraccionaria. Estos tipos son fundamentales en cálculos de alta precisión.

Rust implementa el estándar IEEE 754 para aritmética de punto flotante, garantizando compatibilidad con otros lenguajes y sistemas, y proporcionando un comportamiento predecible en diferentes plataformas.

Tipos Disponibles

Rust proporciona dos tipos de punto flotante que difieren en su precisión y rango de representación:

Tipo	Precisión	Rango aproximado
`f32`	6–9 dígitos significativos	$\pm 3.4 \times 10^{38}$
`f64`	15–17 dígitos significativos	$\pm 1.8 \times 10^{308}$

Tabla 1: Características de los tipos de punto flotante según IEEE 754

Precisión simple `f32`: Utiliza 32 bits para almacenar el valor. Es más eficiente en términos de memoria y puede ofrecer ventajas de rendimiento.

Precisión doble `f64`: Es el tipo predeterminado en Rust debido a que los procesadores modernos operan con `f64` a velocidades comparables a `f32`, mientras que proporciona significativamente mayor precisión.

```
let pi_aproximado = 3.14159265359;     // f64 predeterminado
let euler: f32 = 2.71828;              // f32 explícito
let gravedad: f64 = 9.80665;           // f64 explícito
```

Inferencia de Tipos

Cuando no se especifica el tipo, Rust infiere `f64` como predeterminado para literales con punto decimal:

```
let x = 3.14;              // Tipo inferido: f64
let y = 2.5;               // Tipo inferido: f64
let resultado = x * y;     // f64
```

Para forzar el uso de `f32`, se debe especificar mediante anotación de tipo o sufijo:

```
let a: f32 = 3.14;         // Anotación de tipo
let b = 2.5f32;            // Sufijo de tipo
```

Limitaciones de Precisión

> **Advertencia importante:** Los números de punto flotante no pueden representar todos los valores decimales con exactitud absoluta. Esta limitación es inherente a la representación binaria utilizada por el estándar IEEE 754 y afecta a todos los lenguajes de programación modernos.

Ejemplo clásico de imprecisión:

```
let resultado = 0.1 + 0.2;
println!("{}", resultado);  // Salida: 0.30000000000000004
```

Valores Especiales

El estándar IEEE 754 define valores especiales para representar condiciones excepcionales:

```
// Infinito positivo y negativo
println!("Infinito: {}", f64::INFINITY);
println!("Infinito negativo: {}", f64::NEG_INFINITY);

// Not a Number (NaN)
println!("NaN: {}", f64::NAN);

// Operaciones que producen valores especiales
let div_cero = 1.0 / 0.0;           // INFINITY
let div_cero_neg = -1.0 / 0.0;      // NEG_INFINITY
let raiz_negativa = (-1.0_f64).sqrt();  // NaN
```

Verificación de Valores Especiales

Rust proporciona métodos para detectar estos casos:

```
let valor = 1.0 / 0.0;

println!("¿Es infinito?: {}", valor.is_infinite());
println!("¿Es finito?: {}", valor.is_finite());
println!("¿Es NaN?: {}", valor.is_nan());
println!("¿Es signo negativo?: {}", valor.is_sign_negative());
println!("¿Es signo positivo?: {}", valor.is_sign_positive());
```

Constantes y Límites

Todos los tipos de punto flotante proporcionan constantes útiles:

```
// Límites numéricos
println!("f32 min: {}, max: {}", f32::MIN, f32::MAX);
// -3.4028235e38  3.4028235e38
println!("f64 min: {}, max: {}", f64::MIN, f64::MAX);
// -1.7976931348623157e308  1.7976931348623157e308
```

Operaciones Matemáticas

Rust proporciona una amplia colección de métodos matemáticos para tipos de punto flotante:

Redondeo

```
let valor = 3.7;

println!("floor (hacia abajo): {}", valor.floor());     // 3.0
println!("ceil (hacia arriba): {}", valor.ceil());      // 4.0
println!("round (más cercano): {}", valor.round());     // 4.0
println!("trunc (parte entera): {}", valor.trunc());    // 3.0
```

Operaciones con Signo

```
let negativo = -15.8;

println!("Valor absoluto: {}", negativo.abs());         // 15.8
println!("Signum (+1, 0 o -1): {}", negativo.signum()); // -1.0
println!("Copiar signo: {}", 10.0_f64.copysign(negativo)); // -10.0
```

Potencias y Raíces

```
let base = 4.0;

println!("Potencia entera: {}", base.powi(3));          // 64.0
```

```rust
println!("Potencia decimal: {}", base.powf(1.5));       // 8.0
println!("Raíz cuadrada: {}", base.sqrt());             // 2.0
println!("Raíz cúbica: {}", 27.0_f64.cbrt());           // 3.0
```

Descomposición de Valores

```rust
let numero:f32 = 42.75;

println!("Parte entera: {}", numero.trunc());           // 42.0
println!("Parte fraccionaria: {}", numero.fract());     // 0.75
```

Formato de Salida

Rust proporciona especificadores de formato para controlar la presentación de números de punto flotante:

```rust
let valor = 123.456789;

// Controlar decimales
println!("2 decimales: {:.2}", valor);          // 123.46
println!("5 decimales: {:.5}", valor);          // 123.45679

// Notación científica
println!("Científica minúscula: {:e}", valor);  // 1.23456789e2
println!("Científica mayúscula: {:E}", valor);  // 1.23456789E2

// Ancho y alineación
println!("Ancho 10, 2 dec: {:10.2}", valor);    // "    123.46"
println!("Alineado izq: {:<10.2}", valor);      // "123.46    "
```

Conversiones entre Tipos

Al igual que con los enteros, las conversiones entre tipos de punto flotante deben ser explícitas:

```rust
let x: f64 = 3.14159265359;

let y: f32 = x as f32;
// Conversión de f64 a f32 pérdida de precisión
// 3.1415927

let a: f32 = 42.5;

let b: f64 = a as f64; // Conversión de f32 a f64 sin pérdida

// Conversión a enteros trunca la parte decimal
let entero: i32 = x as i32; // 3
```