Tuplas

En Rust, una tupla es un tipo compuesto que permite agrupar varios valores dentro de un único contenedor. Su tamaño es fijo: una vez creada, no puede crecer ni reducirse.

Una tupla es útil cuando:

- Quieres empaquetar distintos tipos de datos.
- No necesitas asignar nombre a cada campo.
- Quieres retornar varios valores desde una función.
- Deseas manipular datos agrupados de manera temporal y ligera.

Rust las define como `(valor1, valor2, valor3, ...)`.

Características principales

- **Puede contener valores de tipos distintos.**
  `(i32, f64, &str)`
- **El orden importa.**
  `(1, "Hola")` es diferente de `("Hola", 1)`.
- **Su tamaño es fijo.**
  No se pueden agregar ni remover elementos.
- **Son ligeras y rápidas.**
  Las tuplas viven en la stack cuando es posible, por lo que no requieren asignación dinámica.
- **Tipos heterogéneos.**
  A diferencia de los arrays, las tuplas pueden contener elementos de diferentes tipos en una misma estructura.

Creación de tuplas

Se definen entre paréntesis:

```
let persona: (&str, i32, bool) = ("Luis", 24, true);
```

Rust también puede inferir los tipos:

```
let persona = ("Luis", 24, true);  // (&str, i32, bool)
```

Acceso por índice

Cada valor dentro de la tupla se accede usando un índice con punto '.'

Los índices empiezan en 0.

```
println!("Nombre: {}", persona.0);  // elemento 1: Luis
println!("Edad: {}", persona.1);    // elemento 2: 24
println!("Activa: {}", persona.2);  // elemento 3: true
```

Desestructuración

Desempaqueta la tupla en variables:

```
let persona = ("Luis", 25, true);
let (nombre, edad, activo) = persona;

println!("{}", nombre);   // Luis
```

```
println!("{}", edad);     // 25
println!("{}", activo);   // true
```

Ignorar valores con '_'

```
let persona = ("Luis", 25, true);
let (_, edad, _) = persona;

println!("{}", edad);  // 25
```

Debug en tuplas

Puedes imprimir una tupla con:

```
println!("{:?}", persona);  // ("Luis", 24, true)
```

Siempre que todos sus elementos implementen `Debug`.

Mutabilidad

La tupla completa debe ser mutable para modificar uno de sus campos.

Ejemplo:

```
let mut persona = ("Luis", 25, true);
persona.0 = "Ana";
persona.1 = 30;
persona.2 = false;

println!("{:?}", persona);
```

Resultado:

```
("Ana", 30, false)
```

Tupla vacía

Rust tiene una tupla especial: la tupla vacía '`()`'.

También se llama:

- unit type
- unit value

```
fn saludo() {
    println!("Hola!");
}

fn main() {
    let x = saludo();
    println!("{:?}", x);  // imprime ()
}
```

Cuando una función no retorna nada, en realidad retorna Unit type.

Rest pattern

Puedes usar '`..`' para ignorar múltiples elementos intermedios:

```
let tupla = (1, 2, 3, 4, 5);
let (primero, .., ultimo) = tupla;
```

```rust
println!("primero = {}, ultimo = {}", primero, ultimo);
// primero = 1, ultimo = 5
```

```rust
let tupla = (1, 2, 3, 4, 5);
let (primero, segundo, ..) = tupla;

println!("{}, {}", primero, segundo);  // 1, 2
```

Tuplas anidadas

Puedes combinar tuplas dentro de otras:

```rust
let anidada = ((1, 2), (3, 4));

println!("{}", (anidada.0).1);  // 2
println!("{}", (anidada.1).0);  // 3
```

Para mayor claridad, puedes desestructurar:

```rust
let anidada = ((1, 2), (3, 4));
let ((a, b), (c, d)) = anidada;

println!("a={}, b={}, c={}, d={}", a, b, c, d);
// a=1, b=2, c=3, d=4
```

Tuplas con un solo elemento

Esto confunde a muchos:

```rust
let x = (5);    // esto NO es una tupla, solo es un i32
let y = (5,);   // esto SÍ es una tupla de un elemento
```

Las tuplas de un solo elemento deben llevar coma final.

Comparaciones

Las tuplas se pueden comparar si todos sus elementos implementan los traits necesarios:

```rust
fn main() {
    let t1 = (1, 2);
    let t2 = (1, 3);

    println!("{}", t1 < t2);   // true
    println!("{}", t1 == t2);  // false
}
```

La comparación se hace elemento por elemento, de izquierda a derecha (orden lexicográfico).

Límite de elementos

Rust implementa automáticamente traits como `Debug`, `Clone`, `PartialEq`, etc., para tuplas de hasta **12 elementos**.

```rust
// Esto funciona sin problema
let t = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12);
println!("{:?}", t);
```

Para tuplas con más de 12 elementos, necesitarás implementar manualmente estos traits si los necesitas:

```rust
// Esto compila, pero Debug no está implementado automáticamente
let t = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13);
// println!("{:?}", t);  // Error: Debug no implementado
```