

Use for, while, and loop expressions:

Los programas suelen tener bloques de código que deben repetirse in situ. Podemos usar expresiones de bucle para indicar al programa cómo realizar las repeticiones. Para imprimir todas las entradas de una agenda de teléfonos, podemos usar una expresión de bucle a fin de indicar al programa cómo imprimir desde la primera hasta la última entrada.

Rust ofrece tres expresiones de bucle para hacer que un programa repita un bloque de código:

- `loop`: se repite, a menos que se produzca una detención manual.
- `while`: se repite mientras una condición permanezca en `true`.
- `for`: se repite para todos los valores de una recopilación.


En esta unidad, echaremos un vistazo a cada una de estas expresiones de bucle.

Que no pare el bucle:

La expresión `loop` crea un bucle infinito. Esta palabra clave nos permite repetir las acciones en el cuerpo de expresiones de forma continua. Las acciones se repiten hasta que realizamos alguna acción directa para detener el bucle.

En el ejemplo siguiente se imprime el texto "We loop forever!". y no se detiene por sí solo. La acción `println!` lo continua repitiendo.

Rust


 Copiar

```
loop {  
    println!("We loop forever!");  
}
```

Cuando se utiliza una expresión **loop**, la única manera de detener el bucle es si, como programador, interviene directamente. Puede agregar código específico para detener el bucle o escribir una instrucción de teclado, como Ctrl+C, a fin de parar la ejecución del programa.

La manera más común de detener una expresión **loop** es usando la palabra clave **break** para establecer un punto de interrupción:

Rust

 Copiar

```
loop {  
    // Keep printing, printing, printing...  
    println!("We loop forever!");  
    // On the other hand, maybe we should stop!  
    break;  
}
```

Cuando el programa encuentra la palabra clave **break**, detiene la ejecución de las acciones en el cuerpo de la expresión **loop** y continúa con la instrucción de código siguiente.

La palabra clave **break** revela una característica especial de la expresión **loop**. Con la palabra clave **break**, puede detener la repetición de las acciones en el cuerpo de expresiones y también devolver un valor en el punto de interrupción.

En el ejemplo siguiente se muestra cómo podemos usar la palabra clave **break** en una expresión **loop** para devolver también un valor:

```
Rust Copiar

let mut counter = 1;
// stop_loop is set when loop stops
let stop_loop = loop {
    counter *= 2;
    if counter > 100 {
        // Stop loop, return counter value
        break counter;
    }
};
// Loop should break when counter = 128
println!("Break the loop at counter = {}.", stop_loop);
```

Este es el resultado:

```
Rust Copiar

Break the loop at counter = 128.
```

El cuerpo de nuestra expresión **loop** realiza estas acciones sucesivas:

1. Declara la variable **stop_loop**.
2. Indica al programa que enlace el valor de la variable al resultado de la expresión **loop**.
3. Inicia el bucle. Ejecuta las acciones en el cuerpo de la expresión **loop**:

Cuerpo del bucle:

4. A: Incrementa el valor de **counter** para que sea el doble del valor actual.

B: Comprueba el valor de counter.

C: Si el valor de **counter** es mayor que 100:
Se para el bucle y se devuelve el valor de **counter**.

D: Si el valor de counter no es mayor que 100:
Repite las acciones del cuerpo del bucle.

5. Establece el valor de `stop_loop` en el valor de `counter`, que es el resultado de la expresión `loop`.

El cuerpo de expresiones **loop** puede tener más de un punto de interrupción. Cuando la expresión tiene varios puntos de interrupción, cada uno de ellos debe devolver el mismo tipo de valor. Todos los valores deben ser de tipo entero, de cadena, o `bool`, y así sucesivamente. Cuando un punto de interrupción no devuelve un valor de forma explícita, el programa interpreta el resultado de la expresión como una **tupla vacía, ()**.

Bucle "while":

El **bucle while** usa una expresión condicional. El bucle se repite siempre que la expresión condicional siga siendo **true**. Esta palabra clave nos permite ejecutar las acciones en el cuerpo de expresiones hasta que la expresión condicional sea `false`.

Un **bucle while** comienza evaluando una expresión condicional booleana. Si la expresión condicional se evalúa como **true**, se ejecutan las acciones del cuerpo. Una vez completadas las acciones, el control vuelve a la expresión condicional. Cuando la expresión condicional se evalúa como `false`, se detiene la expresión `while`.

En el ejemplo siguiente se imprime el texto "We loop a while ...". Cada repetición del bucle prueba la condición "**el recuento es menor que 5**". Mientras la condición siga siendo verdadera, se ejecutaran las acciones en el cuerpo de expresiones. Una vez que la condición ya no sea verdadera, el **bucle while** se detiene y el programa continúa con la instrucción de código siguiente.

```
while counter < 5 {  
    println!("We loop a while...");  
    counter = counter + 1;  
}
```

Bucle "for" para estos valores:

El **bucle for** utiliza un iterador para procesar una colección de elementos. El bucle repite las acciones en el cuerpo de expresiones para cada elemento de la colección.

Este tipo de repetición de bucle se denomina **iteración**.

Cuando se completan todas las iteraciones, el bucle se detiene.

En Rust, podemos **"iterar"** cualquier tipo de colección, como una matriz, un vector o un mapa hash. Rust utiliza un **iterador** para desplazarse por todos y cada uno de los elementos de la colección, del primero al último.

El **bucle for** utiliza una variable temporal como **iterador**. La variable se declara de forma implícita al principio de la expresión del bucle y el valor actual se establece con cada iteración.

En el código siguiente, la colección es la matriz `big_birds` y el iterador se denomina `bird`.

```
let big_birds = ["ostrich", "peacock", "stork"];  
for bird in big_birds
```

Se accede a los elementos de la colección mediante el método `iter()`. La expresión `for` enlaza el valor actual del iterador al resultado del método `iter()`. En el cuerpo de expresiones, podemos trabajar con el valor del iterador.

Rust

 Copiar

```
let big_birds = ["ostrich", "peacock", "stork"];
for bird in big_birds.iter() {
    println!("The {} is a big bird.", bird);
}
```

Este es el resultado:

Resultados

 Copiar

```
The ostrich is a big bird.
The peacock is a big bird.
The stork is a big bird.
```

Otra manera sencilla de crear un iterador es usar la notación de intervalo **a..b**. El iterador comienza en el valor **a** y continúa hasta el **b**, en incrementos de uno, pero no usa el valor **b**.

Rust

 Copiar

```
for number in 0..5 {
    println!("{}", number * 2);
}
```

Este código recorre en iteración los números 0, 1, 2, 3 y 4. Enlaza el valor a la variable **number** a cada iteración del bucle.

Este es el resultado:

Resultados

 Copiar

```
0
2
4
6
8
```