Ciclos en Rust

NACION CRYPTO By DECRY

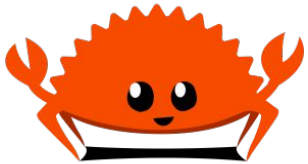
if, else if, else

```
If condicion_1 {
        Bloque_1
} else if condicion_2 {
        Bloque_2
} else {
        Bloque_n
}
```

```rust
let x = 8;

if x == 8 {
    println!("x es ocho!");
} else if x % 4 {
    println!("x es divisible por 4!");
} else {
    println!("x no es ocho y tampoco divisible por
4");
}
```

Cada condición debe ser una expresión de tipo booleano.

*Rust no convierte implícitamente numeros ó punteros a valores booleanos.

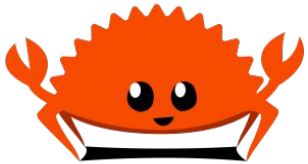


Let if

```
If let patron = condicion_1 {
        Bloque_1
} else {
        Bloque_2
}
```

```
let x = 5;
let y = if x == 5 {
  10
} else {
15
};
```

El patron obtendra el valor del bloque_1 si la condición_1 se cumple, sino el valor del bloque_2



Loop

```
loop {
        Bloque
}
```

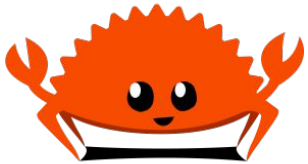
```
'etiqueta: loop {
        Instrucciones a ejecutar  loop {
        If condicion { //tipo booleano
                break etiqueta;
        }
}
```

Hay dos palabras reservadas para romper ó continuar el ciclo:

break: terminar el ciclo

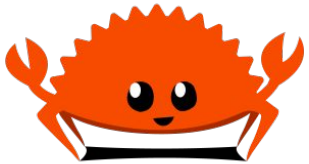
continue: continuar a la siguiente iteración

```
let mut x = 0;

loop {
        x +=1;
        println! ("El valor de x es {},x);

        If x == 10 { break}
}
```



Loop + etiqueta

```rust
// uso de loop con etiqueta de break
{
let mut count: i32 = 0;
'counting_up: loop {
    println!("Contador = {count}");
    let mut remaining: i32 = 10;
    loop {
        println!("Faltante = {remaining}");
        if remaining == 9 {
            break;
        }
        if count == 2 {
            break 'counting_up;
        }
        remaining -= 1;
        }
    count += 1;
    }
    println!("Contador final = {count}");
}
```

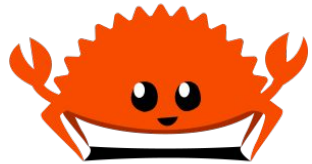


while

```
While condition {
        Bloque
}
```

```
let mut numero = 3;

while numero != 0 {
        println!("{numero}!");

        numero -= 1;

}
```

La función `.enumerate()`, nos permite contar el número de iteraciones.



for

```
for patron in iterador {
        Bloque
}
```

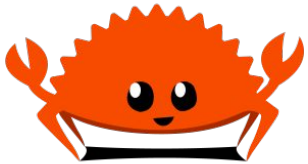
For en lenguaje C

```
for ( int i=0; i < 20; i++) {
        printf (“%d “, i);
}
```

En Rust

```
for i in 0..20 {
        Println! (“{ }”, i);
}
```

El operador **..** produce un rango, es un estructura con dos campos inicio y fin.
0..20 es lo mismo que std::ops::Range { start:0, end:20 }



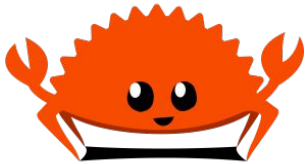
Match

```
match valor {
        Patron  => expression,
        …
}
```

```
match code {
        0 =>  println!("OK"),
        1 =>  println!("cables con problemas"),
        2 =>  println!("Usuario desconectado"),
        _ =>  println!("Error desconocido")
}
```

```
match params.get ("name") {
        some(name) => println!("hola",name),
        None => println!("Usuario desconocido")
}
```

Las expresiones match son como las sentencias **‘switch’** de C pero más flexibles.
_ wildcard para expresar todo.



BIBLIOGRAFIA:

- Programming Rust, Fast, safe systems development - 2nd edition, Jim blandy, Jason Orendorff and Leonora F. S. Tindall
- https://doc.rust-lang.org/book/
- https://polkadothub.io/rust/0-presentaci%C3%B3n/0-presentation
- https://www.youtube.com/watch?v=dad1NQdjd0I&t=619s