# Rust 2023

clase 2

# Temario

- Estructuras de control
- Funciones

# Estructuras de control

# Estructuras de control: if, if-else

```
if condicion_booleana {
        //hace algo porque condicion_booleana es true
}
```

```
if condicion_booleana {
        //hace algo porque la condicion_booleana es true
}else{
        // hace algo porque la condicion_booleana es false
}
```

# Estructuras de control: if-else if

```
if condicion_booleana {
        //hace algo porque la condicion_booleana es true
    }else if otra_condicion{
        // hace algo porque otra_condicion es true
    }else{
        //hace algo porque otra_condicion y condicion_boleana son
false
    }
```

# Estructuras de control:  if con declaración let

```
let data = if condicion_booleana{ 20 } else { 0};
```

```
fn main() {
    let number: i32 = 10;
    let condicion_booleana: bool = number < 10;
    let data: i32 = if condicion_booleana{
        //pueden haber mas intrucciones
        println!("entro por aca!");
        number*number
        } else {
            let mut n: i32 = number;
            n *=2;
            n
        };
    println!("{}", data);
}
```

# Estructuras de control:  match

la forma de match es la siguiente:

```
match algun_valor {
        patron_que_cumple_algun_valor => //hace algo porque lo cumple,
        otro_patron => //hace algo porque lo cumple,
}
```

patrón puede ser: `literals, destructured arrays, enums, structs, tuples, variables, wildcards, placeholders`

# Estructuras de control:  match(con variables)

```
let number = 10;
match number {
    3 => println!("es tres o hace algo porque es 3"),
    7 => println!("es siete o hace algo porque es 7"),
    other => println!("hace algo porque porque no es 3 ni 7"),
}
```

# Estructuras de control:  match(variables-placeholder)

```
let number = 10;
match number {
    3 => println!("es tres o hace algo porque es 3"),
    7 => println!("es siete o hace algo porque es 7"),
    _ => println!("hace algo porque porque no es 3 ni 7"),
}
```

# Estructuras de control: match

```
let number = 10;
match number {
    3 => println!("es tres o hace algo porque es 3"),
    7 => println!("es siete o hace algo porque es 7"),
    _ => (),
}
```

# Estructuras de control:  loop

```
fn main() {
    let mut number = 10;
    loop{
        number+=1;
        if number == 30{
            break;
        }
    }
    println!("{}", number);
}
```

# Estructuras de control:  loop

```
fn main() {
    let mut number = 10;
    let termina = loop{
        number+=1;
        if number == 30{
            break true
        }
    };
    println!("{} {}", number, termina);
}
```

# Estructuras de control:  loop con tag

```
let mut count = 0;
'counting_up: loop {
    let mut remaining = 10;
    loop {
        if remaining == 9 {
            break;
        }
        if count == 2 {
            break 'counting_up;
        }
        remaining -= 1;
    }
    count += 1;
}
println!("End count = {count}");
```

# Estructuras de control:  while

```
let mut number = 0;
while number < 10{
    println!("{number}");
    number +=2;
};
```

# Estructuras de control: for

```
let arreglo = [1, 2, 3, 4, 5];
for elemento in arreglo {
    println!("el valor es: {elemento}");
}
```

# Estructuras de control:  for

```
let limite = 5;
for i in 1..limite+1 {
    println!("el valor es: {i}");
}
for i in (1..limite+1).rev() {
    println!("el valor es: {i}");
}
```

# Funciones

# Funciones

Como se observó estuvimos viendo una función: main. La definición de una función se realiza con la palabra reservada “fn” a continuación el nombre de la misma (snake case) y luego entre los paréntesis los argumentos. Entre las llaves el código propio del scope de la función.

```
fn mi_nueva_funcion(arg1: tipo, arg2: tipo, arg_n:tipo){

        //codigo propio del scope de la función

}
```

# Funciones

```rust
fn mi_funcion( data:i32){
    println!("{data}");
}
```

```rust
fn mi_funcion( data:[i32; 7]){
    for i in data{
        println!("{i}");
    }
}
```

# Funciones: retornado valores

```rust
fn mi_funcion( data:i32) -> i32{
    println!("{data}");
    return data
}
```

```rust
fn mi_funcion( data:i32) -> i32{
    println!("{data}");
    data
}
```