Traits

Trait ¿Qué es?

- Es una funcionalidad particular que tiene un tipo y puede compartir con otros tipos.
- Podemos usar traits para definir comportamiento de manera abstracta.
- Se pueden usar traits como límites en tipos de datos genéricos para determinada funcionalidad que el tipo genérico debe cumplir.
- Son similares a las interfaces llamadas en otros lenguajes pero con algunas diferencias.

Trait: ejemplo I

```
pub trait MulI32 {
    fn mul(&self, other:i32) -> f64;// abstracto
    fn hace_algo_concreto(&self){// por defecto
        println!("hace_algo_concreto");
    }
}
```

Trait : ejemplo I

```
impl MulI32 for f64{
    fn mul(&self, other:i32) -> f64{
        self * other as f64
    }
}
fn main(){
    let v1 = 2.8;
    let v2 = 4;
    let r = v1.mul(v2);
    println!("{}", r);
}
```

Trait : ejemplo II

```
struct Perro{}
struct Gato{}
fn main(){
    let gato = Gato{};
    let perro = Perro{};
    println!("{} {}", gato.hablar(), perro.hablar());
}
```

Trait : ejemplo II

```
pub trait Animal{
    fn hablar(&self) -> String;
}
impl Animal for Perro{
    fn hablar(&self) -> String{
        "Guau!".to_string()
    }
}
impl Animal for Gato{
    fn hablar(&self) -> String{
        "Miau!".to_string()
    }
}
```

Trait : limitando un generic

```
pub fn imprimir_hablar<T: Animal>(animal: &T) {
    println!("Hablo! {}", animal.hablar());
}
fn main(){
    let gato = Gato{};
    let perro = Perro{};
    imprimir_hablar(&gato);
    imprimir_hablar(&perro);
}
```

Trait : como parámetro

```
pub fn imprimir_hablar(animal1: &impl Animal, animal2: &impl Animal) {
    println!("Hablando! {} {}", animal1.hablar(), animal2.hablar());
}
fn main(){
    let gato = Gato{};
    let perro = Perro{};
    imprimir_hablar(&gato, &perro);
}
```

Trait : como parámetro

```
pub fn imprimir_hablar(animal: &impl Animal) {
    println!("Hablo! {}", animal.hablar());
}
fn main(){
    let gato = Gato{};
    let perro = Perro{};
    imprimir_hablar(&gato);
    imprimir_hablar(&perro);
}
```

Trait : múltiples

```
pub fn imprimir_hablar(animal: &(impl Animal + OtroTrait)) {
    println!("Hablo! {}", animal.hablar());
}
fn main(){
    let gato = Gato{};
    imprimir_hablar(&gato);
}
```

Trait : múltiples con where

```
pub fn imprimir_hablar<T>(animal: &T)
where
    T: Animal + OtroTrait
{
    println!("Hablo! {}", animal.hablar());
}
fn main(){
    let gato = Gato{};
    imprimir_hablar(&gato);
}
```