Ownership

O ingrediente secreto

Como?

	
Gerenciamento automático de memória	Sem coletor de lixo
Sem data races	Paralelismo
Abstrações	De custo zero

Tipos simples são Copy

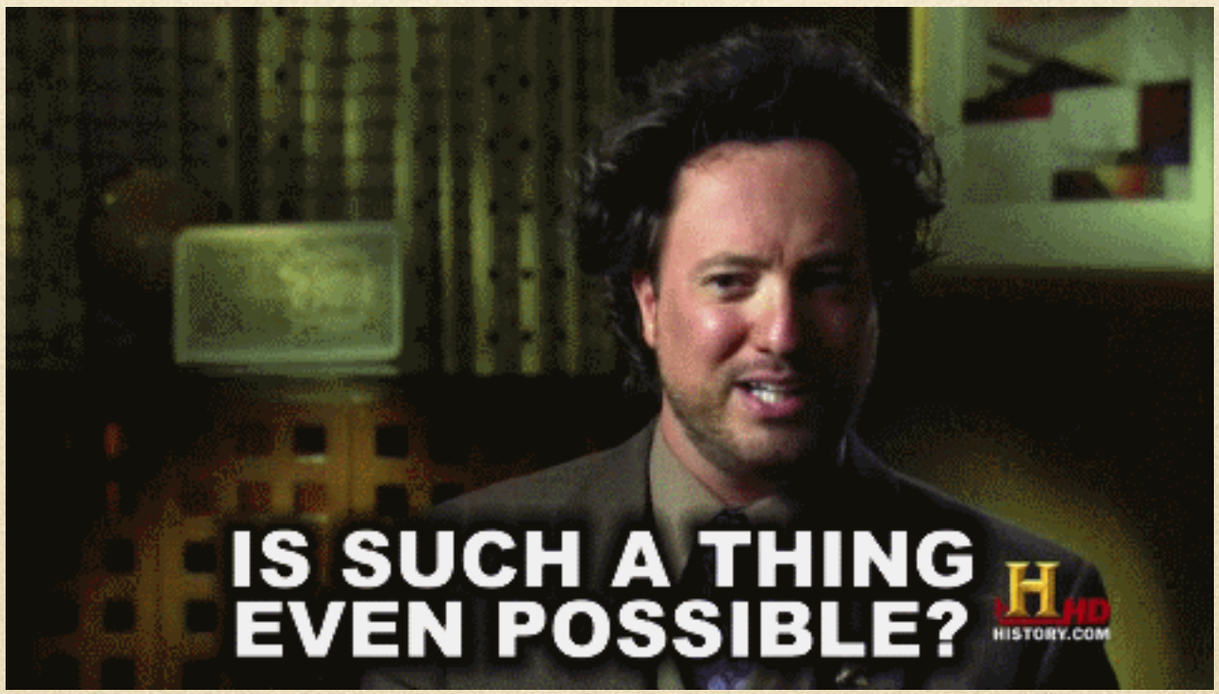
```
fn main() {
    let x = 1;
    foo(x);
}
```

E tipos que precisam ser liberados?

```
fn main() {
    let x = vec![1, 2, 3];
    foo(x);
}
```

Gerenciamento automático de memória sem coletor de lixo

- O compilador sabe exatamente onde a memória deixa de ser acessada e pode ser liberada.



Ownership

- Ao atribuir um valor a uma variável ela se torna a única dona daquele valor.
- No momento em que um valor fica sem dono, ele é liberado.

```
fn foo() {
    let x = vec![1];
}
```

Ownership

- É possível mover a posse de um valor.
- Não é possível utilizar uma variável que não possui um valor.

Ownership

```
fn main() {
    let x = vec![1, 2, 3];
    let w = foo(x);
}

fn foo(v : Vec<i32>) -> Vec<i32> {
    let y = vec![3, 2, 1];
    y
}
```

Ownership

```
fn main() {
    let x = vec![1, 2, 3];
    let w = foo(x);
}

fn foo(v : Vec<i32>) -> Vec<i32> {
    let y = vec![3, 2, 1];
    y
}
```

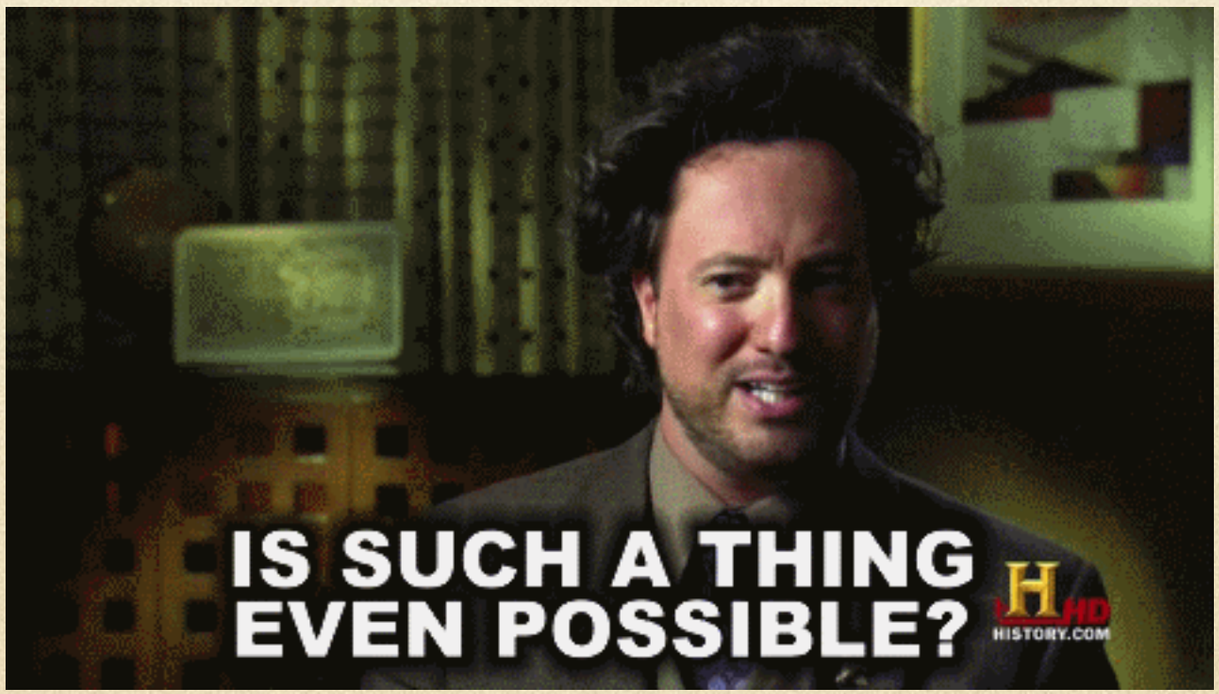
&x

- Lifetime é o escopo no qual um valor é válido.
- O lifetime de uma referência não pode ser maior que o lifetime do valor ao qual ela se refere.

```
fn main() {
    let mut x = &0;
    if *x == 0 {
        let um = 1;
        x = &um; // `um` does not
                 live long enough.
    }
    x + 1;
}
```

Gerenciamento automático de memória sem coletor de lixo

- O compilador sabe exatamente onde a memória deixa de ser acessada e pode liberada.



Paralelismo sem data races

- Se houver uma referência mutável para um valor, ela é o único modo de acessar o valor.

```
fn main() {
    let mut x = 5;
    let mut y = &mut x;
    foo(&x); // Proibido.
}
```


Rust
EVANGELISM
STRIKE FORCE