```gleam
// Composição de funções
// Recebe duas funções e as encadeia para formar uma função que recebe
// a entrada da primeira e retorna a saída da segunda
pub fn compose(fun1: fn(a) -> b, fun2: fn(b) -> c) -> fn(a) -> c {
  fn(a) { fun2(fun1(a)) }
}

pub fn add_one_and_double() {
  let add_one = fn(x: Int) {x + 1}
  let double = fn(x: Int) {x * 2}

  compose(add_one, double)
}
```

```gleam
pub fn add_one_and_double_test() {
  1
  |> gleam_lab.add_one_and_double()
  |> should.equal(4)
}
```

Minha impressões

Gleam busca agregar

Gleam vem com a proposta de agregar e não de substituir. A linguagem tem como um dos principais pilares a interoperabilidade com Erlang e Elixir e pode ser muito bem utilizada em partes críticas de uma aplicação Erlang/Elixir onde seu sistema de tipos seja o diferencial

Também é possível escrever bibliotecas em Gleam para serem utilizadas em aplicações Erlang/Elixir