```gleam
// Funções nomeadas - são criadas utilizando a keyword fn seguido do nome da função
pub fn hello_world() -> String {
  "Hello, from Gleam!"
}

pub fn add(x: Int, y: Int) -> Int {
  x + y
}

pub fn multiply(x: Int, y: Int) -> Int {
  x * y
}

// Funções anônimas - são criadas com a sintaxe fn(argumentos) {corpo da função}
pub fn run() {
  let add = fn(x, y) { x + y }

  add(1, 2)
}

// Funções podem receber como argumento outra função e podem ser encadeadas
pub fn twice(f: fn(t) -> t, x: t) -> t {
  f(f(x))
}

pub fn add_one(x: Int) -> Int {
  x + 1
}

pub fn add_two(x: Int) -> Int {
  twice(add_one, x)
}
```

```gleam
import gleam_lab
import gleam/should

pub fn hello_world_test() {
  gleam_lab.hello_world()
  |> should.equal("Hello, from Gleam!")
}

pub fn add_test() {
  gleam_lab.add(2, 4)
  |> should.equal(6)
}

pub fn multiply_test() {
  gleam_lab.multiply(2, 4)
  |> should.equal(8)
}

pub fn add_two_test() {
  gleam_lab.add_two(1)
  |> should.equal(3)

  gleam_lab.add_two(2)
  |> should.equal(4)
}
```

```erlang
-module(gleam_lab).
-compile(no_auto_import).

-export([hello_world/0, add/2, multiply/2, run/0, twice/2, add_one/1, add_two/1]).

hello_world() ->
    <<"Hello, from Gleam!"/utf8>>.

add(X, Y) ->
    X + Y.

multiply(X, Y) ->
    X * Y.

run() ->
    Add = fun(X, Y) -> X + Y end,
    Add(1, 2).

twice(F, X) ->
    F(F(X)).

add_one(X) ->
    X + 1.

add_two(X) ->
    twice(fun add_one/1, X).
```

Código Erlang gerado a partir da compilação