```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

```gleam
import gleam/int
import gleam/string

// Gleam suporta tipos opacos, que permitem exportar o tipo sem exportar seu detalhes internos
pub opaque type Counter {
    Counter(value: Int, step: Int)
}

// Exemplo de tagged union (ou sum type) - o tipo Message possui 3 diferentes construtores
pub type Message {
    Increment
    Decrement
    Reset
}

// A função init recebe como argumento apenas o parametro inteiro step e retorna um Counter
pub fn init(step: Int) -> Counter {
    Counter(value: 0, step: step)
}

// Na função update nós vamos fazer o handle das mensages e casar a mensagem com a sua função específica
pub fn update(counter: Counter, message: Message) -> Counter {
    case message {
        Increment -> Counter(..counter, value: counter.value + counter.step)
        Decrement -> Counter(..counter, value: counter.value - counter.step)
        Reset -> init(counter.step)
    }
}

// E por fim a função render recebe um Counter e retorna uma string com um label e o valor desse counter
pub fn render(counter: Counter) -> String {
    string.append("The current count is: ", int.to_string(counter.value))
}
```