```gleam
// Curry
// Recebe uma função com aridade 2 e retorna o seu equivalente transformado
pub fn curry2(fun: fn(a, b) -> value) {
  fn(a) { fn(b) { fun(a, b) } }
}

// Gleam suporta nativamente curry com até aridade 6, sendo facilmente expandivel se necessário
pub fn curry3(fun: fn(a, b, c) -> value) {
  fn(a) { fn(b) { fn(c) { fun(a, b, c) } } }
}

pub fn curry4(fun: fn(a, b, c, d) -> value) {
  fn(a) { fn(b) { fn(c) { fn(d) { fun(a, b, c, d) } } } }
}

pub fn curry5(fun: fn(a, b, c, d, e) -> value) {
  fn(a) { fn(b) { fn(c) { fn(d) { fn(e) { fun(a, b, c, d, e) } } } } }
}

pub fn curry6(fun: fn(a, b, c, d, e, f) -> value) {
  fn(a) {
    fn(b) { fn(c) { fn(d) { fn(e) { fn(f) { fun(a, b, c, d, e, f) } } } } }
  }
}

// Meu curry com aridade 7
pub fn curry7(fun: fn(a, b, c, d, e, f, g) -> value) {
  fn(a) {
    fn(b) { fn(c) { fn(d) { fn(e) { fn(f) { fn(g) { fun(a, b, c, d, e, f, g) } } } } } }
  }
}
```

```gleam
pub fn curry2_test() {
  let fun = fn(a, b) { a + b }
  let curried = gleam_lab.curry2(fun)

  curried(1)(2)
  |> should.equal(3)
}

pub fn curry3_test() {
  let fun = fn(a, b, c) { a + b - c}
  let curried = gleam_lab.curry3(fun)

  curried(1)(2)(3)
  |> should.equal(0)
}

pub fn curry4_test() {
  let fun = fn(a, b, c, d) { a + b + c + d}
  let curried = gleam_lab.curry4(fun)

  curried(1)(1)(1)(1)
  |> should.equal(4)
}

// Teste para o meu curry de aridade 7
pub fn curry7_test() {
  let fun = fn(a, b, c, d, e, f, g) { a + b + c + d + e + f + g}
  let curried = gleam_lab.curry7(fun)

  curried(1)(1)(1)(1)(1)(1)(1)
  |> should.equal(7)
}
```