Funções e Structs

Pontos

```
struct Point {
  var x:Double
  var y:Double

  func distancia(de outro:Point) -> Double {
    var diffX = self.x - outro.x
    var diffY = self.y - outro.y
    var sumSqs = diffX*diffX + diffY*diffY
    return sqrt(sumSqs)
  }
  func igual(a outro:Point) -> Bool {
    return self.x == outro.x && self.y == outro.y
  }
}
```

Usando Pontos, versão 1

```
var pt1 = Point(x:10, y:10)
var pt2 = Point(x:5, y:5)

// 7.071067811
pt1.distancia(de:pt2)
// false
pt1.igual(a:pt2)
```

Pontos com as funções fora do struct

```
struct Point {
  var x:Double
  var y:Double
}
func distancia(entre um: Point, e outro:Point) -> Double {
  var diffX = um.x - outro.x
  var diffY = um.y - outro.y
  var sumSqs = diffX*diffX + diffY*diffY
  return sqrt(sumSqs)
}
func iguais(_ um:Point, _ outro:Point) -> Bool {
  return um.x == outro.x && um.y == outro.y
}
```

Usando Pontos, versão 2

```
var pt1 = Point(x:10, y:10)
var pt2 = Point(x:5, y:5)

// 7.071067811
distancia(entre:pt1, e:pt2)
// false
iguais(pt1, pt2)
```

Nomes importam!!!

```
func distancia(entre um: Point, e outro:Point) -> Double {...}
func iguais(_ um:Point, _ outro:Point) -> Bool {...}
```

Versus

```
struct Point {
  func distancia(de outro:Point) -> Double {...}
  func igual(a outro:Point) -> Bool {...}
}
```

Movendo um ponto

```
struct Point {
  var x:Double
  var y:Double

  func mover(dx:Double, dy: Double) {
    x = x + dx
    y = y + dy
  }
}


let ponto : Point = Point(x: 10, y: 20)
ponto.mover(dx: 1, dy: 2)
```

Não é possível apenas atualizar variáveis!

```
struct Point {
  var x:Double
  var y:Double

  mutating func mover(dx:Double, dy: Double) {
    x = x + dx
    y = y + dy
  }
}


var ponto : Point = Point(x: 10, y: 20)
ponto.mover(dx: 1, dy: 2)
```

Inicializadores

Funções especiais (nome `init`, sem tipo de retorno)

Servem para inicializar as propriedades do struct

Permitem que structs sejam criados de diferentes maneiras

Se nenhum for criado, um é definido implicitamente

```
struct Pergunta {
  var pergunta:String
  var resposta:String
  var peso:Int = 1
}
```

```
struct Pergunta {
  var pergunta:String
  var resposta:String
  var peso:Int = 1
  // o inicializador implícito é igual a este
  init(pergunta:String, resposta:String, peso:Int) {
    self.pergunta = pergunta
    self.resposta = resposta
    self.peso = peso
  }
}
```

```
struct Pergunta {
  var pergunta:String
  var resposta:String
  var peso:Int = 1
  // o inicializador implícito é igual a este
  init(pergunta:String, resposta:String, peso:Int) {
    self.pergunta = pergunta
    self.resposta = resposta
    self.peso = peso
  }
  // inicializador extra, sem pontos
  init(pergunta:String, resposta:String) {
    self.pergunta = pergunta
    self.resposta = resposta
  }
}
```

Defina um struct `Retangulo` que guarda os dois lados de um retângulo e um `Ponto` correspondente ao seu centro. Essa struct deve incluir funções para:

- calcular a área do retângulo;
- verificar se dois retângulos são iguais; e
- calcular a distância entre tal retângulo e um ponto qualquer.

Computed Properties

```
struct Retangulo {
  var lado1:Double
  var lado2:Double
  var area:Double { // computed property
    get {
      return lado1 * lado2
    } // pode ser set também.
  }
}
var rect = Retangulo(lado1: 10, lado2: 20)
rect.area // devolve 200
rect.area = 10 // erro em tempo de compilação
```

Defina um struct `Pokemon` que guarda algumas informações relevantes de um pokemon:

- nome
- tipo (um ou dois)
- pontos de HP totais
- pontos de HP atuais
- uma lista de ataques, que incluem um nome e um dano básico

Um pokemon deve ser capaz de verificar se outro pokemon tem um tipo em comum com ele e deve ser capaz de atacar outro pokemon com um determinado ataque, infligindo dano no pokemon atacado.