Funções como Valores e Clausuras

Funções são valores em Swift!

```
func pitagoras(x:Double, y:Double) ->
    Double {
  return sqrt(x*x + y*y)
}
// guarda a função na variável f
var f = pitagoras
f(3,4) // devolve 5
```

```
func pitagoras(x:Double, y:Double) ->
    Double {
  return sqrt(x*x + y*y)
}
// guarda a função na variável f
var f:(Double, Double)->Double = pitagoras
f(3,4) // devolve 5
```

Funções sem resultado ou parâmetros também têm tipos!

```
func funcaoVazia() {
  print("Sem parâmetros ou resultado")
}
var g:() -> Void = funcaoVazia
g()
```

Se funções são valores, podem ser usadas em outras funções?

Clausuras

Blocos de código que podem ser passados para funções e devolvidos por elas

- Ou seja, são **valores**

São, elas próprias, funções

Funções tradicionais são só um caso especial

Podem ser nomeadas ou **anônimas**

Exemplo: ordenação

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

func criterio(x:Int, y:Int) -> Bool {
   return x > y
}

var numerosOrdenados = numeros.sorted(by: criterio)
//imprime [456, 345, 321, 236, 58, 45, 34, 22, 17]
print(numerosOrdenados)
```

Exemplo: ordenação

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

func criterio2(x:Int, y:Int) -> Bool {
   return x < y
}

var numerosOrdenados = numeros.sorted(by: criterio2)
//imprime [17, 22, 34, 45, 58, 236, 321, 345, 456]
print(numerosOrdenados)
```

Expressões de clausura

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

var numerosOrdenados = numeros.sorted(by:
  // Função anônima
  { (x:Int, y:Int) -> Bool in return x < y }
)
//imprime [17, 22, 34, 45, 58, 236, 321, 345, 456]
print(numerosOrdenados)
```

Inferência de tipos funciona em expressões de clausura...

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

var numerosOrdenados = numeros.sorted(by:
  // Função anônima
  { x, y in return x < y }
)
//imprime [17, 22, 34, 45, 58, 236, 321, 345, 456]
print(numerosOrdenados)
```

...e o return torna-se desnecessário em clausuras de uma linha

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

var numerosOrdenados = numeros.sorted(by:
  // Função anônima
  { x, y in x < y }
)
//imprime [17, 22, 34, 45, 58, 236, 321, 345, 456]
print(numerosOrdenados)
```

E, neste cenário, nomes de parâmetros também

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

var numerosOrdenados = numeros.sorted(by:
  // Função anônima
  { $0 < $1 }
)
//imprime [17, 22, 34, 45, 58, 236, 321, 345, 456]
print(numerosOrdenados)
```

Operadores também são funções!

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

var numerosOrdenados = numeros.sorted(by: <)



//imprime [17, 22, 34, 45, 58, 236, 321, 345, 456]
print(numerosOrdenados)
```

Operadores também são funções!

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

var numerosOrdenados = numeros.sorted(by: >)



//imprime [456, 345, 321, 236, 58, 45, 34, 22, 17]
print(numerosOrdenados)
```

Trailing closures

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

var numerosOrdenados = numeros.sorted() { $0 < $1 }

//imprime [17, 22, 34, 45, 58, 236, 321, 345, 456]
print(numerosOrdenados)
```

Trailing closures

```
var numeros:[Int] = [456,34,236,45,321,58,345,22,17]

var numerosOrdenados = numeros.sorted { $0 < $1 }

//imprime [17, 22, 34, 45, 58, 236, 321, 345, 456]
print(numerosOrdenados)
```

Outras funções de alta ordem

```
map<T>(_ transform: (Element) throws -> T)
    rethrows -> [T]

flatMap<ElementOfResult>(_ transform: (Element)
    throws -> ElementOfResult?)
    rethrows -> [ElementOfResult]

func filter(_ isIncluded: (Element) throws -> Bool)
    rethrows -> [Element]

func reduce<Result>(_ initialResult: Result,
    _ nextPartialResult: (Result, Element) throws -> Result)
    rethrows -> Result
```

Captura de Variáveis

```
func criarContador() -> (() -> Int){
  var count = 0

  return { () -> Int in
    count = count + 1
    return count
  }
}
```

Captura de Variáveis

```
func criarContador() -> (() -> Int){
  var count = 0

  return { () -> Int in
    count = count + 1
    return count
  }
}

var contador = criarContador()
```

Captura de Variáveis

```
func criarContador() -> (() -> Int){
  var count = 0

  return { () -> Int in
    count = count + 1
    return count
  }
}

var contador = criarContador()
contador() // devolve 1
```

Captura de Variáveis

```
func criarContador() -> (() -> Int){
  var count = 0

  return { () -> Int in
    count = count + 1
    return count
  }
}

var contador = criarContador()
contador() // devolve 1
contador() // devolve 2
contador() // devolve 3...
```