```
-- *************  Tuplas
-- Valores combinados em uma dupla, tripla, quadrupla, ou ....
(2,3)
(2, "ab")
(4, "cd")
('a', 3, "ab", [1,2,3])
```

```
-- Do ponto do vista de tipos, uma tuplas têm:
--  - número fixo de elementos
--  - elementos podem ter tipos diferentes

(2,3) :: (Int, Int)
(2, "ab") :: (Int, String)
(4, "cd") :: (Int, String)
('a', 3, "ab", [1,2,3]) :: (Char, Int, String, [Int])
```

```
-- Podemos usar tuplas em funções que devolvem vários valores. Por
exemplo

minMax :: Int -> Int -> (Int, Int)
minMax a b
  | a >= b    = (b, a)
  | otherwise = (a, b)
```

```
-- Funções seletoras de duplas
fst (3, 0) --> 3
snd (3, 0) --> 0
```

```
-- Uma função que soma os dois elementos de uma dupla
somaD :: (Int, Int) -> Int
somaD d = fst d + snd d

> soma (4, 2)
6
```

```
-- Ao invés de usar seletoras podemos usar *casamento de padrões*

somaD :: (Int, Int) -> Int
somaD (a,b) = a + b

-- Uma dupla ou dois argumentos?
-- Observe e compare as duas definições
soma :: Int -> Int -> Int
soma a b = a+b
```

```
-- fst e snd são definidas no Prelude usando
-- casamento de padrões

fst (a, _) = a

snd (_, b) = b
```

```
-- Recordando que casamento de padrões permite
-- que usemos várias equações para uma definição
em :: (Float, Float) -> String
em (0, 0) = "origem"
em (_, 0) = "abcissa"
em (0, _) = "ordenada"
em (_, _) = "outro"
```

```
-- Casamento de padrões em definições locais
maxMin :: Int -> Int -> (Int, Int)
maxMin a b = (ma, mi)
  where (mi, ma) = minMax a b
```

```
-- Padrões podem ser aninhados
```

```
shift ((Int, Int), Int) -> (Int, (Int, Int))
shift ((a,  b), c) = (a, (b, c))
```

```
-- Tuplas são estrutras úteis para modelagem de dados

-- Exemplo

-- Itens de compra
("Sal", 1.4)
("Açucar",  5.7)
("Agua Sanitária", 2.5)

-- Carrinho de compras representado por uma lista de itens
[("Sal", 1.4), ("Açucar",  5.7), ("Sal", 1.4)]


-- Casamento de padrões no gerador de uma compreensão
total :: [(String, Float)]  -> Float
total itens = sum [ pr | (_, pr) <- itens]
```

```
-- *** Nomeando tipos
type ItemCompra = (String, Float)
type CarrinhoCompras = [ItemCompra]

preco :: ItemCompra -> Float
preco (_, p) = p

total :: CarrinhoCompras  -> Float
total itens = sum [ preco item | item <- itens]

-- *type* define apenas sinônimos (não cria um tipo novo)
```

```
-- Ex. Queremos modelar Vetores 2D.
-- Uso listas ou duplas?
```

```
-- Exercícios de fixação
-- Defina funções para
```

```
-- . Dado um carrinho de compras, retornar a lista dos
--   nomes dos itens mais caros
-- . Dado um carrinho e o nome de um item, retornar o total
--   no carrinho considerando somente itens com o nome dado.
totalItem :: String -> CarrinhoCompras -> Float
totalItem nm cs = sum [ pr | (nm1, pr) <- cs, nm == nm1 ]

-- . O total gasto com itens cujo valor unitário é inferior ou
--   ou igual a 5
-- . O total gasto com itens cujo valor unitário ultrapassa
--   100
```