

Funções e Operadores Comuns em Haskell

Operadores

```
-- Operadores Aritméticos
3 + 4          → 7
10 - 3         → 7
2 * 5          → 10
9 / 2          → 4.5
9 `div` 2       → 4
9 `mod` 2       → 1
2 ^ 3           → 8
2 ** 3          → 8.0
negate 5        → -5

-- Operadores de Comparação
3 == 3          → True
3 /= 4          → True
2 < 5           → True
6 > 3           → True
3 <= 3          → True
4 >= 5          → False

-- Operadores Lógicos
not True        → False
True && False    → False
True || False    → True

-- Operadores de Composição e Aplicação
((*) . (+)) 4    → 14
sum $ [1,2,3,4]   → 10

-- Outros Operadores
[10,20,30] !! 1   → 20
1 : [2,3]          → [1,2,3]
[1,2] ++ [3,4]     → [1,2,3,4]
[1..5]             → [1,2,3,4,5]
(\x -> x + 1) 5    → 6
```

Funções Básicas (Sem Listas)

```
-- Funções Numéricas
abs (-5)          → 5
signum (-8)        → -1
negate 10          → -10
succ 4             → 5
pred 4             → 3
max 3 7            → 7
min 3 7            → 3
div 10 3            → 3
mod 10 3            → 1
10 `div` 3          → 3
10 `mod` 3          → 1
quot 10 3           → 3
rem 10 3            → 1

-- Funções Matemáticas
sqrt 9             → 3.0
exp 1               → 2.7182818
log 100              → 4.605170
sin (pi/2)           → 1.0
cos 0                → 1.0
tan (pi/4)           → 1.0
round 3.7             → 4
floor 3.7             → 3
ceiling 3.1            → 4
fromIntegral 5 + 2.5   → 7.5

-- Funções Lógicas e Comparação
not True            → False
```

```

True && False          → False
True || False           → True
3 == 3                 → True
3 /= 4                 → True
5 > 3                  → True
5 < 3                  → False
5 >= 5                 → True
5 <= 2                 → False

-- Controle e Estruturas
if 5 > 3 then "sim" else "não" → "sim"
case x of 1 -> "um"; 2 -> "dois"; _ -> "outro"
let x = 5 in x + 3        → 8
-- 'where' pode definir variáveis ao final de uma função

-- Conversões e Tipos
fromIntegral (length [1,2,3]) / 2 → 1.5
show 123                  → "123"
read "45" :: Int           → 45

-- Outras Funções Úteis
error "falha!"            → erro de execução
undefined                  → erro de execução
seq 1 2                    → 2
fst (10,20)                → 10
snd (10,20)                → 20

```

Funções com Listas

```

-- Listas e transformação
map (+1) [1,2,3]          → [2,3,4]
filter even [1,2,3,4,5,6]   → [2,4,6]
zip [1,2,3] ['a','b','c']  → [(1,'a'),(2,'b'),(3,'c')]
zipWith (+) [1,2,3] [4,5,6] → [5,7,9]
concat [[1,2],[3,4]]       → [1,2,3,4]
reverse [1,2,3]            → [3,2,1]
take 3 [10,20,30,40,50]   → [10,20,30]
drop 2 [1,2,3,4,5]         → [3,4,5]
takeWhile (<5) [1,2,3,6,7] → [1,2,3]
dropWhile (<5) [1,2,3,6,7] → [6,7]

-- Redução e agregação
foldl (+) 0 [1,2,3,4]      → 10
foldr (:) [] [1,2,3]        → [1,2,3]
sum [1,2,3,4]               → 10
product [1,2,3,4]           → 24
and [True, True, False]     → False
or [True, False, False]     → True
any even [1,3,4,7]           → True
all odd [1,3,5]              → True
maximum [4,9,2,7]            → 9
minimum [4,9,2,7]            → 2

-- Acesso e informações
head [5,6,7]                → 5
last [5,6,7]                 → 7
init [1,2,3,4]               → [1,2,3]
tail [1,2,3,4]               → [2,3,4]
length [1,2,3,4]              → 4
elem 3 [1,2,3,4]             → True
notElem 5 [1,2,3,4]           → True
null []                      → True

-- Geração e repetição
iterate (*2) 1 !! 5          → 32
repeat 7 !! 3                 → 7
replicate 3 "Hi"              → ["Hi","Hi","Hi"]
take 10 (cycle [1,2])         → [1,2,1,2,1,2,1,2,1,2]
enumFromTo 1 5                 → [1,2,3,4,5]

-- Strings
words "ola mundo haskell"    → ["ola","mundo","haskell"]
unwords ["ola","mundo"]        → "ola mundo"
lines "a\nb\nc"                → ["a","b","c"]

```

```
unlines ["a", "b", "c"]           → "a\nb\nc\n"  
-- Conversão  
show [1,2,3]                     → "[1,2,3]"  
read "42" :: Int                  → 42  
  
-- Tuplas e funções de ordem superior  
fst (1,"a")                      → 1  
snd (1,"a")                      → "a"  
curry (\(x,y) -> x + y) 2 3      → 5  
uncurry (+) (2,3)                 → 5  
id 10                            → 10  
const 42 "abc"  
flip (/) 2 10                     → 5.0  
(.) (*2) (+3) 4                  → 14  -- composição: (*2)●(+3)  
($) sum [1,2,3,4]                 → 10  -- aplicação: sum $ [1,2,3,4]
```