

Próximo:[Operações de arredondamento](#), Anterior:[Conversões numéricas](#), Acima:[Números](#) [[Conteúdo](#)][[Índice](#)]

## 3.6 Operações Aritméticas

O Emacs Lisp fornece as quatro operações aritméticas tradicionais (adição, subtração, multiplicação e divisão), bem como funções de resto e módulo, e funções para adicionar ou subtrair 1. Exceto por %, cada uma dessas funções aceita argumentos inteiros e de ponto flutuante, e retorna um número de ponto flutuante se algum argumento for de ponto flutuante.

### Função: 1+ *número ou marcador*

Esta função retorna *número ou marcador* mais 1. Por exemplo,

```
(setq foo 4)
⇒ 4
(1+ foo)
⇒ 5
```

Esta função não é análoga ao operador C ++— ela não incrementa uma variável. Ele apenas calcula uma soma. Assim, se continuarmos,

```
foo
⇒ 4
```

Se você quiser incrementar a variável, você deve usar `setq`, assim:

```
(setq foo (1+ foo))
⇒ 5
```

### Função: 1- *número ou marcador*

Esta função retorna *número ou marcador* menos 1.

### Função: + *&rest números-ou-marcadores*

Esta função adiciona seus argumentos juntos. Quando não há argumentos, +retorna 0.

```
(+)
⇒ 0
(+ 1)
⇒ 1
(+ 1 2 3 4)
⇒ 10
```

### Função: - *&número-ou-marcador opcional &repouso mais-números-ou-marcadores*

A - função serve a dois propósitos: negação e subtração. Quando - tem um único argumento, o valor é o negativo do argumento. Quando há vários argumentos, - subtrai cada um dos *mais números ou marcadores* de *number-or-marker*, cumulativamente. Se não houver argumentos, o resultado é 0.

```
(- 10 1 2 3 4)
  ⇒ 0
(- 10)
  ⇒ -10
(-)
  ⇒ 0
```

### Função: *\* &rest números-ou-marcadores*

Essa função multiplica seus argumentos e retorna o produto. Quando não há argumentos, *\** retorna 1.

```
(*)
  ⇒ 1
(* 1)
  ⇒ 1
(* 1 2 3 4)
  ⇒ 24
```

### Função: */ divisores de número e resto*

Com um ou mais *divisores*, esta função divide *number* por cada divisor em *divisores* sucessivamente e retorna o quociente. Sem *divisores*, esta função retorna  $1/\text{número}$ , ou seja, o inverso multiplicativo de *número*. Cada argumento pode ser um número ou um marcador.

Se todos os argumentos forem inteiros, o resultado será um inteiro, obtido pelo arredondamento do quociente para zero após cada divisão.

```
(/ 6 2)
  ⇒ 3
(/ 5 2)
  ⇒ 2
(/ 5,0 2)
  ⇒ 2,5
(/ 5 2,0)
  ⇒ 2,5
(/ 5,0 2,0)
  ⇒ 2,5
(/ 4,0)
  ⇒ 0,25
(/ 4)
  ⇒ 0
(/ 25 3 2)
  ⇒ 4
(/ -17 6)
  ⇒ -2
```

Se você dividir um inteiro pelo inteiro 0, o Emacs sinaliza um `arith-error` (consulte [Erros](#)). A divisão de ponto flutuante de um número diferente de zero por zero produz infinito positivo ou negativo (consulte [Float Basics](#)).

### Função: *% divisor de dividendos*

Esta função retorna o resto inteiro após a divisão do *dividendo* pelo *divisor* . Os argumentos devem ser inteiros ou marcadores.

Para quaisquer dois inteiros *dividendo* e *divisor* ,

```
(+ (% divisor de dividendos )
   (* (/ divisor de dividendos ) divisor ))
```

sempre é igual a *dividendo* se o *divisor* for diferente de zero.

```
(% 9 4)
⇒ 1
(% -9 4)
⇒ -1
(% 9 -4)
⇒ 1
(% -9 -4)
⇒ -1
```

### Função: *mod divisor de dividendos*

Esta função retorna o valor do *divisor* do módulo do *dividendo* ; ou seja, o resto após a divisão do *dividendo* pelo *divisor* , mas com o mesmo sinal do *divisor* . Os argumentos devem ser números ou marcadores.

Ao contrário de `%` , `mod` permite argumentos de ponto flutuante; ele arredonda o quociente para baixo (em direção a menos infinito) para um inteiro e usa esse quociente para calcular o resto.

Se o *divisor* for zero, `mod` sinaliza um `arith-error` erro se ambos os argumentos forem inteiros e retorna um NaN caso contrário.

```
(mod 9 4)
⇒ 1
(mod -9 4)
⇒ 3
(mod 9 -4)
⇒ -3
(mod -9 -4)
⇒ -1
(modificação 5.5 2.5)
⇒ .5
```

Para quaisquer dois números *dividendo* e *divisor* ,

```
(+ (mod divisor de dividendos )
   (* ( divisor mínimo de dividendos ) divisor ))
```

sempre é igual a *dividendo* , sujeito a erro de arredondamento se um dos argumentos for ponto flutuante e a um `arith-error` se *dividendo* for um número inteiro e o *divisor* for 0. Para `floor` , consulte [Conversões numéricas](#) .

Próximo: [Operações de arredondamento](#), Anterior: [Conversões numéricas](#), Acima: [Números](#) [[Conteúdo](#)][[Índice](#)]