

Próximo: [Conversões numéricas](#), Anterior: [Predicados em números](#), Acima: [Números](#) [[Conteúdo](#)][[Índice](#)]

3.4 Comparação de Números

Para testar números para igualdade numérica, você normalmente deve usar `=` em vez de predicados de comparação não numérica como `eq`, `eq?` ou `equal`. Objetos distintos de ponto flutuante e inteiros grandes podem ser numericamente iguais. Se você usar `eq` para compará-los, você testa se eles são o mesmo *objeto*; se você usa `eq?` ou `equal`, você testa se seus valores são *indistinguíveis*. Por outro lado, `=` usa comparação numérica e, às vezes, retorna `t` quando uma comparação não numérica retornaria `nil` e vice-versa. Consulte [Noções básicas de flutuação](#).

No Emacs Lisp, se dois `fixnums` são numericamente iguais, eles são o mesmo objeto Lisp. Ou seja, `eq?` é equivalente a `=` on `fixnums`. Às vezes, é conveniente usar `eq` para comparar um valor desconhecido com um `fixnum`, porque `eq` não relata um erro se o valor desconhecido não for um número - ele aceita argumentos de qualquer tipo. Por outro lado, `=` sinaliza um erro se os argumentos não forem números ou marcadores. No entanto, é uma prática de programação melhor usar `=`, se possível, mesmo para comparar números inteiros.

Às vezes é útil comparar números com `eq?` ou `equal`, que tratam dois números como iguais se eles tiverem o mesmo tipo de dados (ambos inteiros ou ambos pontos flutuantes) e o mesmo valor. Por outro lado, `=` pode tratar um número inteiro e um número de ponto flutuante como iguais. Consulte [Predicados de igualdade](#).

Há outro problema: como a aritmética de ponto flutuante não é exata, geralmente é uma má ideia verificar a igualdade dos valores de ponto flutuante. Geralmente é melhor testar a igualdade aproximada. Aqui está uma função para fazer isso:

```
(defvar fuzz-factor 1.0e-6)
(defun aproximadamente igual (xy)
  (ou (= xy)
      (< (/ (abs (-xy))
            (máximo (abs x) (abs y)))
        fator fuzz)))
```

Função: `=` *número-ou-marcador & resto-número-ou-marcadores*

Esta função testa se todos os seus argumentos são numericamente iguais e retorna em `t` caso afirmativo, `nil` caso contrário.

Função: `eq?` *valor1 valor2*

Esta função age como `eq` exceto quando ambos os argumentos são números. Ele compara números por tipo e valor numérico, de modo que `(eq? 1.0 1)` retorna `nil`, mas `(eq? 1.0 1.0)` e `(eq? 1 1)` ambos retornam `t`. Isso pode ser usado para comparar números inteiros grandes, bem como pequenos.

Função: `/=` *número-ou-marcador1 número-ou-marcador2*

Essa função testa se seus argumentos são numericamente iguais e retorna `t` se não forem e `nil` se forem.

Função: `<` *número-ou-marcador & resto-número-ou-marcadores*

Essa função testa se cada argumento é estritamente menor que o argumento a seguir. Retorna tse sim, nil caso contrário.

Função: `<= number-or-marker &rest number-or-markers`

Essa função testa se cada argumento é menor ou igual ao argumento a seguir. Retorna tse sim, nil caso contrário.

Função: `> número-ou-marcador & resto número-ou-marcadores`

Esta função testa se cada argumento é estritamente maior que o argumento a seguir. Retorna tse sim, nil caso contrário.

Função: `>= número-ou-marcador & resto número-ou-marcadores`

Esta função testa se cada argumento é maior ou igual ao argumento a seguir. Retorna tse sim, nil caso contrário.

Função: `número ou marcador máximo e números ou marcadores de descanso`

Esta função retorna o maior de seus argumentos.

```
(máximo 20)
⇒ 20
(máximo 1 2,5)
⇒ 2,5
(máximo 1 3 2,5)
⇒ 3
```

Função: `número mínimo ou marcador e números restantes ou marcadores`

Esta função retorna o menor de seus argumentos.

```
(min -4 1)
⇒ -4
```

Função: `número abs`

Esta função retorna o valor absoluto de *number* .

Próximo: [Conversões numéricas](#), Anterior: [Predicados em números](#), Acima: [Números](#) [[Conteúdo](#)][[Índice](#)]