

Próximo:[Avaliação adiada](#), Anterior:[Aspas](#), Acima:[Avaliação](#) [Conteúdo][Índice]

10.5 Avaliação

Na maioria das vezes, os formulários são avaliados automaticamente, em virtude de sua ocorrência em um programa que está sendo executado. Em raras ocasiões, pode ser necessário escrever um código que avalie um formulário que é calculado em tempo de execução, como após ler um formulário do texto que está sendo editado ou obter um de uma lista de propriedades. Nessas ocasiões, use a `eval` função. Muitas vezes `eval` não é necessário e outra coisa deve ser usada em seu lugar. Por exemplo, para obter o valor de uma variável, enquanto `eval` funciona, `symbol-value` é preferível; ou em vez de armazenar expressões em uma lista de propriedades que precisam passar `eval`, é melhor armazenar funções que são passadas para `funcall`.

As funções e variáveis descritas nesta seção avaliam formulários, especificam limites para o processo de avaliação ou registram valores retornados recentemente. Carregar um arquivo também faz avaliação (consulte [Carregando](#)).

Geralmente é mais limpo e flexível armazenar uma função em uma estrutura de dados e chamá-la com `funcall` ou `apply`, do que armazenar uma expressão na estrutura de dados e avaliá-la. O uso de funções fornece a capacidade de passar informações para elas como argumentos.

Função: forma eval e lexical opcional

Esta é a função básica para avaliar uma expressão. Ele avalia o *formulário* no ambiente atual e retorna o resultado. O tipo do objeto de *formulário* determina como ele é avaliado. Consulte [Formulários](#) .

O argumento *léxico* especifica a regra de escopo para variáveis locais (consulte [Variable Scoping](#)). Se for omitido ou `nil`, isso significa avaliar o *formulário* usando a regra de escopo dinâmico padrão. Se for `t`, isso significa usar a regra de escopo lexical. O valor de *léxico* também pode ser um alista não vazio especificando um *ambiente léxico* específico para ligações léxicas; no entanto, esse recurso é útil apenas para fins especializados, como em depuradores Emacs Lisp. Consulte [Ligação léxica](#) .

Como `eval` é uma função, a expressão de argumento que aparece em uma chamada para `eval` é avaliada duas vezes: uma vez como preparação antes de ser chamada e novamente pela `eval` própria função. Aqui está um exemplo:

```
(setq foo 'bar)
      ⇒ barra
(setq bar 'baz)
      ⇒ baz
;; Aqui eval recebe argumento foo
(eval 'foo)
      ⇒ barra
;; Aqui eval recebe argumento bar, que é o valor de foo
(eval foo)
      ⇒ baz
```

O número de chamadas atuais para `eval` é limitado a `max-lisp-eval-depth` (veja abaixo).

Comando: eval-region start end e função de leitura de fluxo opcional

Esta função avalia os formulários no buffer atual na região definida pelas posições *start* e *end*. Ele lê formulários da região e evalos chama até que o final da região seja alcançado, ou até que um erro seja sinalizado e não tratado.

Por padrão, eval-region não produz nenhuma saída. No entanto, se *stream* for não- nil, qualquer saída produzida por funções de saída (consulte [Funções de saída](#)), bem como os valores resultantes da avaliação das expressões na região são impressos usando *stream*. Consulte [Fluxos de saída](#).

Se *read-function* for non- nil, deve ser uma função, que é usada em vez de reader expressões uma a uma. Essa função é chamada com um argumento, o fluxo para leitura de entrada. Você também pode usar a variável *load-read-function* (consulte [Como os programas carregam](#)) para especificar essa função, mas é mais robusto usar o argumento *da função de leitura*.

eval-region não move ponto. Sempre retorna nil.

Comando: eval-buffer & opcional buffer-or-name stream filename unibyte print

Isso é semelhante a eval-region, mas os argumentos fornecem recursos opcionais diferentes. eval-buffer opera em toda a parte acessível do buffer *buffer-or-name* (consulte [Limitando](#) no Manual do GNU Emacs). *buffer-or-name* pode ser um buffer, um nome de buffer (uma string) ou nil (ou omitido), o que significa usar o buffer atual. *stream* é usado como em eval-region, a menos que *stream* seja nil e imprima não- nil. Nesse caso, os valores resultantes da avaliação das expressões ainda são descartados, mas a saída das funções de saída é impressa na área de eco. *filename* é o nome do arquivo a ser usado paraload-history (consulte [Descarregando](#)), e o padrão é buffer-file-name (consulte [Nome do arquivo de buffer](#)). Se *unibyte* for non- nil, readconverte strings em unibyte sempre que possível.

eval-current-buffer é um alias para este comando.

Opção do usuário: max-lisp-eval-depth

Esta variável define a profundidade máxima permitida nas chamadas para eval, apply, e funcallantes que um erro seja sinalizado (com mensagem de erro "Lisp nesting exceeds max-lisp-eval-depth").

Este limite, com o erro associado quando é excedido, é uma maneira pela qual o Emacs Lisp evita a recursão infinita em uma função mal definida. Se você aumentar max-lisp-eval-depth muito o valor, esse código poderá causar estouro de pilha. Em alguns sistemas, esse estouro pode ser tratado. Nesse caso, a avaliação normal do Lisp é interrompida e o controle é transferido de volta para o loop de comando de nível superior (top-level). Observe que não há como entrar no depurador Emacs Lisp nessa situação. Consulte [Depuração de erros](#).

O limite de profundidade conta os usos internos de eval, apply e funcall, como para chamar as funções mencionadas em expressões Lisp e avaliação recursiva de argumentos de chamada de função e formas de corpo de função, bem como chamadas explícitas em código Lisp.

O valor padrão desta variável é 800. Se você definir um valor menor que 100, Lisp irá redefinir para 100 se o valor fornecido for atingido. A entrada no depurador Lisp aumenta o valor, se houver pouco espaço sobrando, para garantir que o próprio depurador tenha espaço para executar.

max-specpdl-size fornece outro limite no aninhamento. Consulte [Variáveis Locais](#).

Variável: valores

O valor desta variável é uma lista dos valores retornados por todas as expressões que foram lidas, avaliadas e impressas de buffers (incluindo o minibuffer) pelos comandos padrão do Emacs que

fazem isso. (Observe que isso *não* inclui avaliação em *ielm* buffers, nem avaliação usando C-j, C-x C-e, e comandos de avaliação semelhantes em lisp-interaction-mode.) Os elementos são ordenados mais recentes primeiro.

```
(conjunto x 1)
  ⇒ 1
(lista 'A (1+ 2) auto-save-default)
  ⇒ (A 3t)
valores
  ⇒ ((A 3t) 1 ...)
```

Esta variável é útil para fazer referência a valores de formulários avaliados recentemente. Geralmente é uma má ideia imprimir o valor de valuessi mesmo, pois isso pode ser muito longo. Em vez disso, examine elementos específicos, como este:

```
; Consulte o resultado da avaliação mais recente.
(nº 0 valores)
  ⇒ (A 3t)

;; Isso colocou um novo elemento,
;; então todos os elementos retrocedem um.
(nº 1 valores)
  ⇒ (A 3t)

;; Isso obtém o elemento que estava próximo ao mais recente
;; antes deste exemplo.
(nº 3 valores)
  ⇒ 1
```

Próximo:[Avaliação adiada](#), Anterior:[Aspas](#), Acima:[Avaliação](#) [Conteúdo][Índice]