Próximo:Documentação da Função, Anterior:Lambda Simples, Acima:Expressões lambda [Conteúdo][Índice]

13.2.3 Recursos das Listas de Argumentos

Nossa função de exemplo simples, `(lambda (a b c) (+ a b c))`, especifica três variáveis de argumento, portanto, deve ser chamada com três argumentos: se você tentar chamá-la com apenas dois ou quatro argumentos, receberá um `wrong-number-of-argumentserro` (consulte Erros).

Muitas vezes é conveniente escrever uma função que permite que certos argumentos sejam omitidos. Por exemplo, a função `substring` aceita três argumentos—uma string, o índice inicial e o índice final—mas o terceiro argumento assume como padrão o *comprimento* da string se você o omitir. Também é conveniente que certas funções aceitem um número indefinido de argumentos, como as funções `list` e +fazem.

Para especificar argumentos opcionais que podem ser omitidos quando uma função é chamada, basta incluir a palavra-chave `&optional`antes dos argumentos opcionais. Para especificar uma lista de zero ou mais argumentos extras, inclua a palavra-chave `&rest`antes de um argumento final.

Assim, a sintaxe completa para uma lista de argumentos é a seguinte:

```
( vars-obrigatórias …
  [ &opcional [ vars-opcionais … ] ]
 [ &rest [ var-resto ] ] )
```

Os colchetes indicam que as cláusulas `&optional`and `&rest` e as variáveis que as seguem são opcionais.

Uma chamada para a função requer um argumento real para cada um dos *required-vars* . Pode haver argumentos reais para zero ou mais de *optional-vars* e não pode haver nenhum argumento real além disso, a menos que a lista lambda use `&rest`. Nesse caso, pode haver qualquer número de argumentos reais extras.

Se os argumentos reais para as variáveis opcionais e rest forem omitidos, eles sempre serão padronizados como `nil`. Não há como a função distinguir entre um argumento explícito de `nil`e um argumento omitido. No entanto, o corpo da função é livre para considerar `nil`uma abreviação de algum outro valor significativo. Isto é o que `substring`faz; `nil`como o terceiro argumento para `substring`significa usar o comprimento da string fornecida.

> **Nota do Common Lisp:** O Common Lisp permite que a função especifique qual valor padrão usar quando um argumento opcional é omitido; O Emacs Lisp sempre usa arquivos `nil`. O Emacs Lisp não suporta `supplied-p` variáveis que informam se um argumento foi explicitamente passado.

Por exemplo, uma lista de argumentos que se parece com isso:

```
(ab &cd opcional &rest e)
```

binds `a`e `b`aos dois primeiros argumentos reais, que são necessários. Se mais um ou dois argumentos forem fornecidos `c`e `d`estiverem vinculados a eles, respectivamente; quaisquer argumentos após os quatro primeiros são coletados em uma lista e `e`vinculados a essa lista. Assim, se houver apenas dois argumentos, `c`, `d`e `e` são `nil`; se dois ou três argumentos, `d`e `e`são `nil`; se quatro argumentos ou menos, `e`é `nil`. Observe

que exatamente cinco argumentos com um `nil`argumento explícito fornecido `e`farão com que esse `nil`argumento seja passado como uma lista com um elemento, `(nil)`, como com qualquer outro valor único para `e`.

Não há como ter argumentos obrigatórios após os opcionais - não faria sentido. Para ver por que isso deve ser assim, suponha que `c`no exemplo fossem opcionais e `d`obrigatórios. Suponha que três argumentos reais sejam dados; para qual variável seria o terceiro argumento? Seria usado para o *c* , ou para *d* ? Pode-se argumentar para ambas as possibilidades. Da mesma forma, não faz sentido ter mais argumentos (obrigatórios ou opcionais) após um `&rest`argumento.

Aqui estão alguns exemplos de listas de argumentos e chamadas apropriadas:

```
(funcall (lambda (n) (1+ n)) ; Um necessário:
         1) ; requer exatamente um argumento.
     ⇒ 2
(funcall (lambda (n &opcional n1) ; Um obrigatório e um opcional:
           (if n1 (+ n n1) (1+ n))) ; 1 ou 2 argumentos.
         1 2)
     ⇒ 3
(funcall (lambda (n &rest ns) ; Um necessário e um descanso:
           (+ n (apply '+ ns))) ; 1 ou mais argumentos.
         1 2 3 4 5)
     ⇒ 15
```

Próximo:Documentação da Função, Anterior:Lambda Simples, Acima:Expressões lambda [Conteúdo][Índice]