

Próximo:[Portando Conselhos Antigos](#), Anterior:[Aconselhando Funções Nomeadas](#), Acima:

[Funções de aconselhamento](#) [[Conteúdo](#)][[Índice](#)]

### 13.11.3 Formas de redigir conselhos

Aqui estão os diferentes valores possíveis para o argumento *whereadd-function* de *e*, especificando como a *função* *advice-add* de aconselhamento e a função original devem ser compostas.

#### **:before**

Chama a *função* antes da função antiga. Ambas as funções recebem os mesmos argumentos e o valor de retorno da composição é o valor de retorno da função antiga. Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (aplica a função r) (aplica oldfun r))
```

(*add-function* *:before funvar function*) é comparável para ganchos de função única para ganchos normais. (*add-hook 'hookvar function*)

#### **:after**

Chama a *função* após a função antiga. Ambas as funções recebem os mesmos argumentos e o valor de retorno da composição é o valor de retorno da função antiga. Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (progl (aplica oldfun r) (aplica a função r)))
```

(*add-function* *:after funvar function*) é comparável para ganchos de função única para ganchos normais. (*add-hook 'hookvar function 'append*)

#### **:override**

Isso substitui completamente a função antiga pela nova. É claro que a função antiga pode ser recuperada se você chamar *remove-function*.

#### **:around**

Chame *function* em vez da função antiga, mas forneça a função antiga como um argumento extra para *function*. Esta é a composição mais flexível. Por exemplo, ele permite que você chame a função antiga com argumentos diferentes, ou muitas vezes, ou dentro de um *let-binding*, ou às vezes você pode delegar o trabalho para a função antiga e, às vezes, substituí-lo completamente. Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (aplica a função oldfun r))
```

#### **:before-while**

Chame a *função* antes da função antiga e não chame a função antiga se a *função* retornar *nil*. Ambas as funções recebem os mesmos argumentos e o valor de retorno da composição é o valor de retorno da função antiga. Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (e (aplica a função r) (aplica oldfun r)))
```

(add-function :before-while *funvar function*) é comparável para ganchos de função única quando *hookvar* é executado via . (add-hook '*hookvar function*)run-hook-with-args-until-failure

### :before-until

Chame a *função* antes da função antiga e só chame a função antiga se a *função* retornar nil. Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (ou (aplicar função r) (aplicar oldfun r)))
```

(add-function :before-until *funvar function*) é comparável para ganchos de função única quando *hookvar* é executado via . (add-hook '*hookvar function*)run-hook-with-args-until-success

### :after-while

Chame a *função* após a função antiga e somente se a função antiga retornar não- nil. Ambas as funções recebem os mesmos argumentos e o valor de retorno da composição é o valor de retorno de *function* . Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (e (aplica oldfun r) (aplica a função r)))
```

(add-function :after-while *funvar function*) é comparável para ganchos de função única quando *hookvar* é executado via . (add-hook '*hookvar function* 'append)run-hook-with-args-until-failure

### :after-until

Chame a *função* após a função antiga e somente se a função antiga retornar nil. Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (ou (aplicar oldfun r) (aplicar função r)))
```

(add-function :after-until *funvar function*) é comparável para ganchos de função única quando *hookvar* é executado via . (add-hook '*hookvar function* 'append)run-hook-with-args-until-success

### :filter-args

Chame a *função* primeiro e use o resultado (que deve ser uma lista) como os novos argumentos para passar para a função antiga. Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (aplica oldfun (funcall funcall r)))
```

### :filter-return

Chame a função antiga primeiro e passe o resultado para *function* . Mais especificamente, a composição das duas funções se comporta como:

```
(lambda (&rest r) (funcall funcall ( aplicar oldfun r)))
```

Próximo: [Portando Conselhos Antigos](#), Anterior: [Aconselhando Funções Nomeadas](#), Acima:

[Funções de aconselhamento](#) [[Conteúdo](#)][[Índice](#)]