

Próximo:[Combinando Condições](#), Anterior:[Sequenciamento](#), Acima:[Estruturas de controle](#) [Conteúdo][[Índice](#)]

11.2 Condicionais

As estruturas de controle condicional escolhem entre alternativas. O Emacs Lisp tem cinco formas condicionais: `if`, que é praticamente o mesmo que em outras linguagens; `when` `unless`, que são variantes de `if`; `cond`, que é uma instrução case generalizada; e `pcase`, que é uma generalização de `cond` (consulte [Condisional de correspondência de padrões](#)).

Forma Especial: se condição então forma outra forma...

`if` escolhe entre a *forma then* e a forma *else com* base no valor de *condition*. Se a *condição* avaliada for não- `nil`, *então-form* será avaliado e o resultado será retornado. Caso contrário, os *else-forms* são avaliados em ordem textual e o valor do último é retornado. (A *outra* parte de `if` é um exemplo de um `progn`. Veja [Sequenciamento](#).)

Se *condition* tiver o valor `nil`, e nenhuma *outra forma* for fornecida, `if` retornará `nil`.

`if` é uma forma especial porque a ramificação que não é selecionada nunca é avaliada — ela é ignorada. Assim, neste exemplo, `truuenão` é impresso porque `printnunca` é chamado:

```
(se nulo
    (imprima 'verdadeiro)
    'muito falso)
⇒ muito falso
```

Macro: quando a condição então se forma...

Esta é uma variante de `if` onde não há *else-forms*, e possivelmente vários *then-forms*. Em particular,

```
(quando condição a b c)
```

é inteiramente equivalente a

```
(se condição (prog a b c) nil)
```

Macro: a menos que a condição se forme...

Esta é uma variante de `if` onde não há *forma então*:

```
(a menos que a condição a b c)
```

é inteiramente equivalente a

```
(se condição nil
  a b c)
```

Forma especial: cláusula cond ...

condescolhe entre um número arbitrário de alternativas. Cada *cláusula* no cond deve ser uma lista. O CAR desta lista é a *condição*; os elementos restantes, se houver, as *formas-corpo*. Assim, uma cláusula fica assim:

```
( condicionar as formas do corpo ...)
```

condtenta as cláusulas em ordem textual, avaliando a *condição* de cada cláusula. Se o valor de *condition* for non- nil, a cláusula será bem-sucedida; em seguida, condavalia seus *body-forms* e retorna o valor do último dos *body-forms*. Quaisquer cláusulas restantes são ignoradas.

Se o valor de *condition* for nil, a cláusula falha, então condpassa para a cláusula seguinte, tentando sua *condição*.

Uma cláusula também pode ter esta aparência:

```
( condição )
```

Então, se *condition* for non- nil quando testado, o cond formulário retornará o valor de *condition*.

Se todas as *condições* forem avaliadas como nil, de modo que todas as cláusulas falhe, condretornará nil.

O exemplo a seguir tem quatro cláusulas, que testam os casos em que o valor de xé um número, string, buffer e símbolo, respectivamente:

```
(cond ((númerop x) x)
      ((stringp x) x)
      ((buffer x)
       (setq temporário-hack x); várias formas de corpo
       (buffer-name x)); em uma cláusula
      ((símbolo x) (símbolo-valor x)))
```

Muitas vezes queremos executar a última cláusula sempre que nenhuma das cláusulas anteriores foi bem sucedida. Para isso, usamos t como *condição* da última cláusula, assim: . O formulário é avaliado como , que é never , portanto, essa cláusula nunca falha, desde que o gets seja feito. Por exemplo: (t body-forms)ttnilcond

```
(conjunto de 5)
(cond ((eq a 'hack) 'foo)
      (t "padrão"))
→ "padrão"
```

Essa condexpressão retorna foose o valor de a for hacke retorna a string "default"caso contrário.

Qualquer construção condicional pode ser expressa com condou com if. Portanto, a escolha entre eles é uma questão de estilo. Por exemplo:

```
(se a b c )
≡
(cond ( a b ) (t c ))
```

Próximo:[Combinando Condições](#), Anterior:[Sequenciamento](#), Acima:[Estruturas de controle](#) [Conteúdo][[Índice](#)]