

Próximo:[Conversão de String](#), Anterior:[Modificando Strings](#), Acima:[Strings e Personagens](#) [Conteúdo]
[Índice](#)

4.5 Comparação de Caracteres e Strings

Função: `char-igual character1 character2`

Esta função retorna tse os argumentos representarem o mesmo caractere, nilcaso contrário. Esta função ignora diferenças caso `case-fold-search` não seja nil.

```
(char-igual ?x ?x)
  ⇒ t
(deixe ((case-fold-search nil))
  (igual a caracteres ?x ?X))
  ⇒ nada
```

Função: `string= string1 string2`

Esta função retorna tse os caracteres das duas strings corresponderem exatamente. Símbolos também são permitidos como argumentos, neste caso os nomes dos símbolos são usados. O caso é sempre significativo, independentemente de `case-fold-search`.

Esta função é equivalente a `equal` comparar duas strings (veja [Equality Predicates](#)). Em particular, as propriedades de texto das duas strings são ignoradas; use `equal-including-properties` se precisar distinguir entre strings que diferem apenas em suas propriedades de texto. No entanto, ao contrário de `equal`, se um dos argumentos não for uma string ou um símbolo, `string=` sinaliza um erro.

```
(string = "abc" "abc")
  ⇒ t
(string= "abc" "ABC")
  ⇒ nada
(string = "ab" "ABC")
  ⇒ nada
```

Por motivos técnicos, uma string unibyte e multibyte são `equal` se e somente se contiverem a mesma sequência de códigos de caracteres e todos esses códigos estiverem no intervalo de 0 a 127 ([ASCII](#)) ou 160 a 255 ([eight-bit-graphic](#)). No entanto, quando uma string unibyte é convertida em uma string multibyte, todos os caracteres com códigos no intervalo de 160 a 255 são convertidos em caracteres com códigos mais altos, enquanto os caracteres [ASCII](#) permanecem inalterados. Assim, uma string unibyte e sua conversão para multibyte são apenas `equal` a string `for` toda [ASCII](#). Os códigos de caracteres de 160 a 255 não são totalmente adequados em texto multibyte, embora possam ocorrer. Como consequência, a situação em que uma string unibyte e uma string multibyte são `equal` que ambos sejam todos [ASCII](#) é uma estranheza técnica com a qual muito poucos programadores Emacs Lisp são confrontados. Consulte [Representações de texto](#).

Função: `string igual a string1 string2`

`string-equal` é outro nome para `string=`.

Função: string-collate-equalp *string1 string2 &opcional locale ignore-case*

Esta função retorna t se *string1* e *string2* são iguais em relação às regras de agrupamento. Uma regra de agrupamento não é determinada apenas pela ordem lexicográfica dos caracteres contidos em *string1* e *string2*, mas também outras regras sobre as relações entre esses caracteres. Normalmente, ele é definido pelo ambiente de *localidade* com o qual o Emacs está sendo executado.

Por exemplo, caracteres com pontos de codificação diferentes, mas com o mesmo significado, podem ser considerados iguais, como caracteres Unicode com acento grave diferente:

```
(string-collate-equalp (string ?\uFF40) (string ?\u1FEF))
  ⇒ t
```

O argumento opcional *locale*, uma string, substitui a configuração de seu identificador de localidade atual para agrupamento. O valor depende do sistema; um *locale* "en_US.UTF-8" é aplicável em sistemas POSIX, enquanto seria, por exemplo, "enu_USA.1252" em sistemas MS-Windows.

Se *ignore-case* for non- nil, os caracteres serão convertidos em letras minúsculas antes de compará-los.

Para emular o agrupamento compatível com Unicode em sistemas MS-Windows, vincule w32-collate-ignore-punctuation a um valor sem nilvalor, pois a parte do conjunto de códigos da localidade não pode estar "UTF-8" no MS-Windows.

Se o seu sistema não suporta um ambiente de localidade, esta função se comporta como *string-equal*.

Não use esta função para comparar nomes de arquivos para igualdade, pois os sistemas de arquivos geralmente não respeitam a equivalência linguística de strings que o agrupamento implementa .

Função: string< *string1 string2*

Esta função compara duas strings de um caractere por vez. Ele verifica ambas as strings ao mesmo tempo para localizar o primeiro par de caracteres correspondentes que não correspondem. Se o caractere menor desses dois for o caractere de *string1*, *string1* será menor e essa função retornará t. Se o caractere menor for o de *string2*, então *string1* será maior e esta função retornará nil. Se as duas strings corresponderem inteiramente, o valor será nil.

Pares de caracteres são comparados de acordo com seus códigos de caracteres. Lembre-se de que letras minúsculas têm valores numéricos mais altos no conjunto de caracteres ASCII do que suas contrapartes maiúsculas; dígitos e muitos caracteres de pontuação têm um valor numérico menor do que letras maiúsculas. Um caractere ASCII é menor que qualquer caractere não ASCII; um caractere unibyte não ASCII é sempre menor que qualquer caractere multibyte não ASCII (consulte [Representações de texto](#)).

```
(string< "abc" "abd")
  ⇒ t
(string< "abd" "abc")
  ⇒ nada
(string < "123" "abc")
  ⇒ t
```

Quando as strings têm comprimentos diferentes e correspondem ao comprimento de *string1* , o resultado é t. Se eles corresponderem ao comprimento de *string2* , o resultado será nil. Uma string sem caracteres é menor que qualquer outra string.

```
(string< "" "abc")
  ⇒ t
(string< "ab" "abc")
  ⇒ t
(string< "abc" "")
  ⇒ nada
(string< "abc" "ab")
  ⇒ nada
(string< "" "")
  ⇒ nada
```

Símbolos também são permitidos como argumentos, caso em que seus nomes de impressão são comparados.

Função: **string-lessp** *string1 string2*

`string-lessp` é outro nome para `string<`.

Função: **string-maiorp** *string1 string2*

Esta função retorna o resultado da comparação de *string1* e *string2* na ordem oposta, ou seja, é equivalente a chamar `.(string-lessp string2 string1)`

Função: **string-collate-lessp** *string1 string2 &opcional locale ignore-case*

Esta função retorna t se *string1* for menor que *string2* na ordem de agrupamento. Uma ordem de agrupamento não é determinada apenas pela ordem lexicográfica dos caracteres contidos em *string1* e *string2* , mas também regras adicionais sobre as relações entre esses caracteres. Normalmente, ele é definido pelo ambiente de *localidade* com o qual o Emacs está sendo executado.

Por exemplo, caracteres de pontuação e espaço em branco podem ser ignorados para classificação (consulte [Funções de sequência](#)):

```
(ordenar (list "11" "12" "1 1" "1 2" "1.1" "1.2") 'string-collate-lessp)
  ⇒ ("11" "1 1" "1.1" "12" "1 2" "1.2")
```

Esse comportamento depende do sistema; por exemplo, pontuação e espaço em branco nunca são ignorados no Cygwin, independentemente da localidade.

O argumento opcional *locale* , uma string, substitui a configuração de seu identificador de localidade atual para agrupamento. O valor depende do sistema; um *locale* "en_US.UTF-8" é aplicável em sistemas POSIX, enquanto seria, por exemplo, "enu_USA.1252" em sistemas MS-Windows. O valor de *localidade* "POSIX" de ou "C" permite `string-collate-lessp` se comportar como `string-lessp`:

```
(ordenar (listar "11" "12" "1 1" "1 2" "1.1" "1.2")
  (lambda (s1 s2) (string-collate-lessp s1 s2 "POSIX")))
  ⇒ ("1 1" "1 2" "1.1" "1.2" "11" "12")
```

Se *ignore-case* for non- nil, os caracteres serão convertidos em letras minúsculas antes de compará-los.

Para emular o agrupamento compatível com Unicode em sistemas MS-Windows, vincule `w32-collate-ignore-punctuation` a um valor sem nilvalor, pois a parte do conjunto de códigos da localidade não pode estar "UTF-8" no MS-Windows.

Se o seu sistema não suporta um ambiente de localidade, esta função se comporta como `string-lessp`.

Função: string-versão-lessp *string1 string2*

Essa função compara strings lexicograficamente, exceto que trata sequências de caracteres numéricos como se fossem um número de base dez e, em seguida, compara os números. Assim '`foo2.png`' é menor que '`foo12.png`' de acordo com este predicado, mesmo que '`12`' é lexicograficamente "menor" do que '`2`'.

Função: string-prefix-p *string1 string2 &opcional ignore-case*

Esta função retorna non - nil se *string1* é um prefixo de *string2*; ou seja, se *string2* começa com *string1*. Se o argumento opcional *ignore-case* for non- nil, a comparação ignorará as diferenças de maiúsculas e minúsculas.

Função: string-suffix-p *sufixo string &opcional ignore-case*

Esta função retorna non - nil se *sufixo* é um sufixo de *string*; ou seja, se *string* termina com *sufixo*. Se o argumento opcional *ignore-case* for non- nil, a comparação ignorará as diferenças de maiúsculas e minúsculas.

Função: compare-strings *string1 start1 end1 string2 start2 end2 &opcional ignore-case*

Esta função compara uma parte especificada de *string1* com uma parte especificada de *string2*. A parte especificada de *string1* é executada do índice *start1* (inclusive) até o índice *end1* (exclusivo); nil para *start1* significa o início da string, enquanto nil para *end1* significa o comprimento da string. Da mesma forma, a parte especificada de *string2* é executada do índice *start2* até o índice *end2*.

As strings são comparadas pelos valores numéricos de seus caracteres. Por exemplo, *str1* é considerado menor que *str2* se seu primeiro caractere diferente tiver um valor numérico menor. Se *ignore-case* for non- nil, os caracteres serão convertidos em maiúsculas antes de compará-los. Strings unibyte são convertidas em multibyte para comparação (consulte [Representações de texto](#)), de modo que uma string unibyte e sua conversão em multibyte são sempre consideradas iguais.

Se as partes especificadas das duas strings corresponderem, o valor será t. Caso contrário, o valor é um número inteiro que indica quantos caracteres iniciais concordam e qual string é menor. Seu valor absoluto é um mais o número de caracteres que coincidem no início das duas strings. O sinal é negativo se *string1* (ou sua parte especificada) for menor.

Função: string-distance *string1 string2 &opcional bytecompare*

Esta função retorna a *distância Levenshtein* entre a string de origem *string1* e a string de destino *string2*. A distância Levenshtein é o número de alterações de um único caractere — exclusões, inserções ou substituições — necessárias para transformar a string de origem na string de destino; é uma definição possível da *distância de edição* entre strings.

As letras maiúsculas das strings são significativas para a distância calculada, mas suas propriedades de texto são ignoradas. Se o argumento opcional *bytecompare* for non- nil, a função calculará a distância em termos de bytes em vez de caracteres. A comparação por byte usa a representação

interna de caracteres do Emacs, portanto, produzirá resultados imprecisos para strings multibyte que incluem bytes brutos (consulte [Representações de texto](#)); torne as strings unibyte codificando-as (consulte [Codificação Explícita](#)) se precisar de resultados precisos com bytes brutos.

Função: lista de chaves assoc-string & case-fold opcional

Esta função funciona como assoc, exceto que a *chave* deve ser uma string ou símbolo, e a comparação é feita usando compare-strings. Os símbolos são convertidos em strings antes do teste. Se *case-fold* não for nil, *key* e os elementos de *alist* são convertidos em maiúsculas antes da comparação. Ao contrário assocde , essa função também pode corresponder a elementos do alist que são strings ou símbolos em vez de conses. Em particular, *alista* pode ser uma lista de strings ou símbolos em vez de uma lista real. Consulte [Listas de Associação](#).

Veja também a função compare-buffer-substringsem [Comparing Text](#), para uma maneira de comparar texto em buffers. A função string-match, que combina uma expressão regular com uma string, pode ser usada para um tipo de comparação de strings; consulte [Pesquisa Regexp](#).

Próximo:[Conversão de String](#), Anterior:[Modificando Strings](#), Acima:[Strings e Personagens](#) [Conteúdo]
[Índice](#)