Próximo:Strings de formato personalizado, Anterior:Conversão de String, Acima:Strings e Personagens [Conteúdo][Índice]

4.7 Formatando Strings

Formatar significa construir uma string substituindo valores calculados em vários lugares em uma string constante. Essa string constante controla como os outros valores são impressos, bem como onde eles aparecem; é chamado de *string de formato* .

A formatação costuma ser útil para a exibição de mensagens de computação. Na verdade, as funções `message`e `error`fornecem o mesmo recurso de formatação descrito aqui; eles diferem `format-message`apenas em como eles usam o resultado da formatação.

Função: formatar *string &rest objetos*

Esta função retorna uma string igual a *string* , substituindo quaisquer especificações de formato pelas codificações dos *objetos* correspondentes . *Os objetos* de argumentos são os valores computados a serem formatados.

Os caracteres em *string* , além das especificações de formato, são copiados diretamente na saída, incluindo suas propriedades de texto, se houver. Quaisquer propriedades de texto das especificações de formato são copiadas para as representações de string produzidas dos *objetos* de argumento .

A string de saída não precisa ser alocada recentemente. Por exemplo, if `x`é a string `"foo"`, as expressões `(eq x (format x))`e `(eq x (format "%s" x))`podem render `t`.

Função: string de mensagem de formato *e objetos de descanso*

Esta função age como `format`, exceto que também converte quaisquer acentos graves (`` ` ``) e apóstrofos (`'`) em *string* conforme o valor de `text-quoting-style`.

Normalmente, o acento grave e o apóstrofo no formato se traduzem em aspas curvas correspondentes, por exemplo, `"Faltando `%s'"` pode resultar em `"Faltando 'foo'"` . Consulte Estilo de citação de texto , para saber como influenciar ou inibir essa tradução.

Uma especificação de formato é uma sequência de caracteres começando com um '`%`'. Assim, se houver um '`%d`' em *string* , a `format`função o substitui pela representação impressa de um dos valores a serem formatados (um dos *objetos* de argumentos). Por exemplo:

```
(formato "O valor da coluna de preenchimento é %d." coluna de preenchimento)
     ⇒ "O valor da coluna de preenchimento é 72."
```

Uma vez que `format`interpreta '`%`' como especificações de formato, você *nunca* deve passar uma string arbitrária como o primeiro argumento. Isso é particularmente verdadeiro quando a string é gerada por algum código Lisp. A menos que a string seja *conhecida* por nunca incluir nenhum '`%`' caracteres, pass `"%s"`, descritos abaixo, como o primeiro argumento, e a string como o segundo, assim:

```
(formato "%s" string arbitrária )
```

Certas especificações de formato exigem valores de tipos específicos. Se você fornecer um valor que não atende aos requisitos, um erro será sinalizado.

Aqui está uma tabela de especificações de formato válidas:

'%s'

Substitua a especificação pela representação impressa do objeto, feita sem aspas (ou seja, usando `princ`, não `prin1`—consulte Funções de saída). Assim, strings são representadas apenas por seu conteúdo, sem '"' caracteres e símbolos aparecem sem '\' personagens.

Se o objeto for uma string, suas propriedades de texto serão copiadas na saída. As propriedades de texto do '%s' também são copiados, mas os do objeto têm prioridade.

'%S'

Substitua a especificação pela representação impressa do objeto, feita com aspas (ou seja, usando `prin1`—consulte Funções de saída). Assim, as strings são colocadas entre '"' personagens e '\' caracteres aparecem onde necessário antes de caracteres especiais.

'%o'

Substitua a especificação pela representação de base oito de um inteiro. Os inteiros negativos são formatados de maneira dependente da plataforma. O objeto também pode ser um número de ponto flutuante formatado como um inteiro, eliminando qualquer fração.

'%d'

Substitua a especificação pela representação de base dez de um inteiro com sinal. O objeto também pode ser um número de ponto flutuante formatado como um inteiro, eliminando qualquer fração.

'%x'

'%X'

Substitua a especificação pela representação de base dezesseis de um inteiro. Os inteiros negativos são formatados de maneira dependente da plataforma. '%x' usa letras minúsculas e '%X' usa maiúsculas. O objeto também pode ser um número de ponto flutuante formatado como um inteiro, eliminando qualquer fração.

'%c'

Substitua a especificação pelo caractere que é o valor fornecido.

'%e'

Substitua a especificação pela notação exponencial para um número de ponto flutuante.

'%f'

Substitua a especificação pela notação de ponto decimal para um número de ponto flutuante.

'%g'

Substitua a especificação por notação para um número de ponto flutuante, usando notação exponencial ou notação de ponto decimal. A notação exponencial é usada se o expoente for menor que -4 ou maior ou igual à precisão (padrão: 6). Por padrão, os zeros à direita são removidos da

parte fracionária do resultado e um caractere de ponto decimal aparece apenas se for seguido por um dígito.

'%%'

Substitua a especificação por um único '%'. Esta especificação de formato é incomum, pois sua única forma é simples '%%' e que não usa um valor. Por exemplo, `(format "%% %d" 30)`retorna `"% 30"`.

Qualquer outro caractere de formato resulta em um '`Operação de formato inválido`' erro.

Aqui estão vários exemplos, que assumem as `text-quoting-style`configurações típicas:

```
(formato "O valor octal de %d é %o,
         e o valor hexadecimal é %x." 18 18 18)
     ⇒ "O valor octal de 18 é 22,
         e o valor hexadecimal é 12."

(formato-mensagem
 "O nome deste buffer é '%s'." (nome do buffer))
     ⇒ "O nome deste buffer é 'strings.texi'."

(formato-mensagem
 "O objeto buffer é impresso como `%s'." (buffer de corrente))
     ⇒ "O objeto buffer é impresso como 'strings.texi'."
```

Por padrão, as especificações de formato correspondem a valores sucessivos de *objetos* . Assim, a primeira especificação de formato em *string* usa o primeiro desses valores, a segunda especificação de formato usa o segundo desses valores e assim por diante. Quaisquer especificações de formato extra (aquelas para as quais não há valores correspondentes) causam um erro. Quaisquer valores extras a serem formatados são ignorados.

Uma especificação de formato pode ter um *número de campo* , que é um número decimal imediatamente após a inicial '%', seguido por um sinal de dólar literal '$'. Isso faz com que a especificação de formato converta o argumento com o número fornecido em vez do próximo argumento. Os números dos campos começam em 1. Um formato pode conter especificações de formato numeradas ou não numeradas, mas não ambas, exceto que '%%' pode ser misturado com especificações numeradas.

```
(formato "%2$s, %3$s, %%, %1$s" "x" "y" "z")
     ⇒ "y, z, %, x"
```

Depois de '%' e qualquer número de campo, você pode colocar certos *caracteres de bandeira* .

A bandeira '+' insere um sinal de mais antes de um número não negativo, para que sempre tenha um sinal. Um caractere de espaço como sinalizador insere um espaço antes de um número não negativo. (Caso contrário, os números não negativos começam com o primeiro dígito.) Esses sinalizadores são úteis para garantir que os números não negativos e negativos usem o mesmo número de colunas. Eles são ignorados, exceto por '%d', '%e', '%f', '%g', e se ambos os sinalizadores forem usados, '+' tem precedência.

A bandeira '#' especifica uma forma alternativa que depende do formato em uso. Para '%o', garante que o resultado comece com um '0'. Para '%x' e '%X', ele prefixa resultados diferentes de zero com '0x' ou '0X'. Para '%e' e '%f', a '#' sinalizador significa incluir um ponto decimal mesmo se a precisão for zero. Para '%g',

ele sempre inclui um ponto decimal e também força os zeros à direita após o ponto decimal a serem deixados no lugar onde eles seriam removidos.

A bandeira '`0`' garante que o preenchimento consiste em '`0`' caracteres em vez de espaços. Este sinalizador é ignorado para caracteres de especificação não numérica como '`%s`', '`%S`' e '`%c`'. Esses caracteres de especificação aceitam o '`0`' sinalizador, mas ainda preenche com *espaços* .

A bandeira '`-`' faz com que qualquer preenchimento inserido pela largura, se especificado, seja inserido à direita e não à esquerda. Se ambos '`-`' e '`0`' estão presentes, o '`0`' sinalizador é ignorado.

```
(formato "%06d é preenchido à esquerda com zeros" 123)
     ⇒ "000123 é preenchido à esquerda com zeros"

(formato "'%-6d' é preenchido à direita" 123)
     ⇒ "'123' é preenchido à direita"

(formato "A palavra '%-7s' na verdade tem %d letras."
        "foo" (comprimento "foo"))
     ⇒ "A palavra 'foo' na verdade tem 3 letras."
```

Uma especificação pode ter uma *largura* , que é um número decimal que aparece após qualquer número de campo e sinalizadores. Se a representação impressa do objeto contiver menos caracteres do que essa largura, `format`estenda-a com preenchimento. Qualquer preenchimento introduzido pela largura normalmente consiste em espaços inseridos à esquerda:

```
(formato "%5d é preenchido à esquerda com espaços" 123)
     ⇒ "123 é preenchido à esquerda com espaços"
```

Se a largura for muito pequena, `format`não trunca a representação impressa do objeto. Assim, você pode usar uma largura para especificar um espaçamento mínimo entre as colunas sem risco de perda de informações. Nos dois exemplos a seguir, '`%7s`' especifica uma largura mínima de 7. No primeiro caso, a string inserida no lugar de '`%7s`' tem apenas 3 letras e precisa de 4 espaços em branco como preenchimento. No segundo caso, a string `"specification"`tem 13 letras, mas não é truncada.

```
(formato "A palavra '%7s' tem %d letras."
        "foo" (comprimento "foo"))
     ⇒ "A palavra 'foo' tem 3 letras."
(formato "A palavra '%7s' tem %d letras."
        "especificação" (comprimento "especificação"))
     ⇒ "A palavra 'especificação' tem 13 letras."
```

Todos os caracteres de especificação permitem uma precisão opcional após o número do campo, flags e largura, se presentes. A precisão é um ponto decimal '`.`' seguido por uma string de dígitos. Para as especificações de ponto flutuante ('`%e`' e '`%f`'), a precisão especifica quantos dígitos após o ponto decimal devem ser exibidos; se zero, o próprio ponto decimal também é omitido. Para '`%g`', a precisão especifica quantos dígitos significativos mostrar (dígitos significativos são o primeiro dígito antes do ponto decimal e todos os dígitos depois dele). Se a precisão de %g for zero ou não especificada, ela será tratada como 1. Para '`%s`' e '`%S`', a precisão trunca a string para a largura dada, então '`%.3s`' mostra apenas os três primeiros caracteres da representação do *objeto* . Para outros caracteres de especificação, o efeito de precisão é o que as funções de biblioteca local da `printf` família produzem.

Se você planeja usar `read`posteriormente a string formatada para recuperar uma cópia do valor formatado, use uma especificação que permita `read`reconstruir o valor. Para formatar números dessa maneira reversível, você pode usar '`%s`' e '`%S`', para formatar apenas inteiros você também pode usar '`%d`', e para formatar apenas números inteiros não negativos, você também pode usar '`#x%x`' e '`#o%o`'. Outros formatos podem ser problemáticos; por exemplo, '`%d`' e '`%g`' pode manipular NaNs e pode perder precisão e tipo, e '`#x%x`' e '`#o%o`' pode lidar mal com inteiros negativos. Consulte Funções de entrada .

As funções descritas nesta seção aceitam um conjunto fixo de caracteres de especificação. A próxima seção descreve uma função `format-spec`que pode aceitar caracteres de especificação personalizada, como '`%uma`' ou '`%z`'.

Próximo:Strings de formato personalizado, Anterior:Conversão de String, Acima:Strings e Personagens [Conteúdo][Índice]