

7. Entrada e Saída

Existem várias maneiras de apresentar a saída de um programa; os dados podem ser exibidos em forma legível para seres humanos, ou escritos em arquivos para uso posterior. Este capítulo apresentará algumas das possibilidades.

7.1. Refinando a formatação de saída

Até agora vimos duas maneiras de exibir valores: *expressões* e a função `print()`. (Uma outra maneira é utilizar o método `write()` de objetos do tipo arquivo; o arquivo saída padrão pode ser referenciado como `sys.stdout`. Veja a Referência da Biblioteca Python para mais informações sobre isso.)

Muitas vezes se deseja mais controle sobre a formatação da saída do que simplesmente exibir valores separados por espaço. Existem várias maneiras de formatar a saída.

- Para usar **strings literais formatadas**, comece uma string com `f` ou `F`, antes de abrir as aspas ou aspas triplas. Dentro dessa string, pode-se escrever uma expressão Python entre caracteres `{` e `}`, que podem se referir a variáveis, ou valores literais.

```
>>> year = 2016
>>> event = 'Referendum'
>>> f'Results of the {year} {event}'
'Results of the 2016 Referendum'
```

- O método de strings `str.format()` requer mais esforço manual. Ainda será necessário usar `{` e `}` para marcar onde a variável será substituída e pode-se incluir diretivas de formatação detalhadas, mas também precisará incluir a informação a ser formatada.

```
>>> yes_votes = 42_572_654
>>> no_votes = 43_132_495
>>> percentage = yes_votes / (yes_votes + no_votes)
>>> '{:-9} YES votes {:.2%}'.format(yes_votes,
percentage)
' 42572654 YES votes 49.67%'
```

- Finalmente, pode-se fazer todo o tratamento da saída usando as operações de fatiamento e concatenação de strings para criar qualquer layout que se possa imaginar. O tipo string possui alguns métodos que realizam operações úteis para preenchimento de strings para uma determinada largura de coluna.

Quando não é necessário sofisticar a saída, mas apenas exibir algumas variáveis com propósito de depuração, pode-se converter qualquer valor para uma string com as funções `repr()` ou `str()`.

A função `str()` serve para retornar representações de valores que sejam legíveis para as pessoas, enquanto `repr()` é para gerar representações que o interpretador Python consegue ler (ou levantará uma exceção `SyntaxError`, se não houver sintaxe equivalente). Para objetos que não têm uma representação adequada para consumo humano, `str()` devolve o mesmo valor que `repr()`. Muitos valores, tal como números ou estruturas, como listas e dicionários, têm a mesma representação usando quaisquer das funções. Strings, em particular, têm duas representações distintas.

Alguns exemplos:

```
>>> s = 'Hello, world.'
>>> str(s)
'Hello, world.'
>>> repr(s)
"'Hello, world.'"
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'
>>> print(s)
The value of x is 32.5, and y is 40000...
>>> # The repr() of a string adds string quotes and backslashes:
... hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print(hellos)
'hello, world\n'
>>> # The argument to repr() may be any Python object:
... repr((x, y, ('spam', 'eggs'))))
"(32.5, 40000, ('spam', 'eggs'))"
```

O módulo `string` contém uma classe `Template` que oferece ainda outra maneira de substituir valores em strings, usando espaços reservados como `$x` e substituindo-os por valores de um dicionário, mas oferece muito menos controle da formatação.

7.1.1. Strings literais formatadas

Strings literais formatadas (também chamadas f-strings, para abreviar) permite que se inclua o valor de expressões Python dentro de uma string, prefixando-a com `f` ou `F` e escrevendo expressões na forma `{expression}`.

Um especificador opcional de formato pode ser incluído após a expressão. Isso permite maior controle sobre como o valor é formatado. O exemplo a seguir arredonda pi para três casas decimais:

```
>>> import math
>>> print(f'The value of pi is approximately {math.pi:.3f}.')
The value of pi is approximately 3.142.
```

Passando um inteiro após o ':' fará com que o campo tenha um número mínimo de caracteres de largura. Isso é útil para alinhar colunas.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print(f'{name:10} ==> {phone:10d}')
...
Sjoerd      ==>      4127
Jack        ==>      4098
Dcab        ==>      7678
```

Outros modificadores podem ser usados para converter o valor antes de ser formatado. '!a' aplica a função `ascii()`, '!s' aplica a função `str()` e '!r' aplica a função `repr()`

```
>>> animals = 'eels'
>>> print(f'My hovercraft is full of {animals}.')
My hovercraft is full of eels.
>>> print(f'My hovercraft is full of {animals!r}.')
My hovercraft is full of 'eels'.
```

Para uma referência dessas especificações de formatos, veja o guia de referência para [Minilinguagem de especificação de formato](#).

7.1.2. O método `format()`

Um uso básico do método `str.format()` tem esta forma:

```
>>> print('We are the {} who say "{}!".format('knights',
'Ni'))
We are the knights who say "Ni!"
```

As chaves e seus conteúdos (chamados campos de formatação) são substituídos pelos objetos passados para o método `str.format()`. Um número nas chaves pode ser usado para referenciar a posição do objeto passado no método `str.format()`.

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

Se argumentos nomeados são passados para o método `str.format()`, seus valores serão referenciados usando o nome do argumento:

```
>>> print('This {food} is {adjective}.'.format(
...     food='spam', adjective='absolutely horrible'))
This spam is absolutely horrible.
```

Argumentos posicionais e nomeados podem ser combinados à vontade:

```
>>> print('The story of {0}, {1}, and {other}.'.format('Bill',
'Manfred',

other='Georg'))
The story of Bill, Manfred, and Georg.
```

Se uma string de formatação é muito longa, e não se deseja quebrá-la, pode ser bom fazer referência aos valores a serem formatados por nome, em vez de posição. Isto pode ser feito passando um dicionário usando colchetes `[]` para acessar as chaves.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {0[Jack]:d}; Sjoerd: {0[Sjoerd]:d}; '
...     'Dcab: {0[Dcab]:d}'.format(table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

Isto também pode ser feito passando o dicionário como argumento do método, usando a notação `**`.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab:
{Dcab:d}'.format(**table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

Isto é particularmente útil em conjunto com a função embutida `vars()`, que devolve um dicionário contendo todas as variáveis locais.

Como exemplo, as linhas seguintes produzem um conjunto de colunas alinhadas, com alguns inteiros e seus quadrados e cubos:

```
>>> for x in range(1, 11):
```

```
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
1      1      1
2      4      8
3      9     27
4     16     64
5     25    125
6     36    216
7     49    343
8     64    512
9     81    729
10    100   1000
```

Para uma visão completa da formatação de strings com `str.format()`, veja a seção [Sintaxe das strings de formato](#).

7.1.4. Formatação de strings à moda antiga

O operador `%` (módulo) também pode ser usado para formatação de string. Dado `'string' % valores`, as instâncias de `%` em `string` são substituídas por zero ou mais elementos de `valores`. Essa operação é conhecida como interpolação de string. Por exemplo:

```
>>> import math
>>> print('The value of pi is approximately %5.3f.' % math.pi)
The value of pi is approximately 3.142.
```

Mais informação pode ser encontrada na seção [Formatação de String no Formato printf-style](#).

ATIVIDADES

1. **Nome na vertical em escada.** Faça um programa que solicite o nome do usuário e mostre o nome em formato de escada.

```
F
FU
FUL
FULA
FULAN
FULANO
```

2. **Jogo de Forca.** Desenvolva um jogo da forca. O programa terá uma lista de palavras lidas de um arquivo texto e escolherá uma aleatoriamente. O jogador poderá errar 6 vezes antes de ser enforcado.

```
Digite uma letra: A
-> Você errou pela 1ª vez. Tente de novo!

Digite uma letra: O
A palavra é: _ _ _ _ O

Digite uma letra: E
A palavra é: _ E _ _ O

Digite uma letra: S
-> Você errou pela 2ª vez. Tente de novo!
```

7.2. Leitura e escrita de arquivos

A função `open()` devolve um **objeto arquivo**, e é frequentemente usada com dois argumentos: `open(nome_do_arquivo, modo)`.

```
>>> f = open('workfile', 'w')
```

O primeiro argumento é uma string contendo o nome do arquivo. O segundo argumento é outra string, contendo alguns caracteres que descrevem o modo como o arquivo será usado. *modo* pode ser `'r'` quando o arquivo será apenas lido, `'w'` para escrever (se o arquivo já existir seu conteúdo prévio será apagado), e `'a'` para abrir o arquivo para adição; qualquer escrita será adicionada ao final do arquivo. A opção `'r+'` abre o arquivo tanto para leitura como para escrita. O argumento *modo* é opcional, em caso de omissão será assumido `'r'`.

Normalmente, arquivos são abertos em *modo texto*, ou seja, você lê e grava strings, de e para o arquivo, numa codificação específica. Se a codificação não for especificada, o padrão é dependente da plataforma/sistema operacional (consulte `open()`). Incluir `'b'` ao *modo* abre o arquivo em *modo binário*: os dados são lidos e escritos na forma de bytes. Esse modo deve ser usado para todos os arquivos que não contenham texto.

Em modo texto, o padrão durante a leitura é converter terminadores de linha específicos da plataforma (`\n` no Unix, `\r\n` no Windows) para apenas `\n`. Ao escrever no modo de texto, o padrão é converter as ocorrências de `\n` de volta para os finais de linha específicos da plataforma. Essa modificação de bastidores nos dados do arquivo é adequada para arquivos de texto, mas corromperá dados

binários, como arquivos `JPEG` ou `EXE`. Tenha muito cuidado para só usar o modo binário, ao ler e gravar esses arquivos.

É uma boa prática usar a palavra-chave `with` ao lidar com arquivos. A vantagem é que o arquivo é fechado corretamente após o término de sua utilização, mesmo que uma exceção seja levantada em algum momento. Usar `with` também é muito mais curto que escrever seu bloco equivalente `try-finally`:

```
>>> with open('workfile') as f:
...     read_data = f.read()

>>> # We can check that the file has been automatically closed.
>>> f.closed
True
```

Se você não está usando a palavra reservada `with`, então você deveria chamar `f.close()` para fechar o arquivo e imediatamente liberar qualquer recurso do sistema usado por ele.

Aviso

Chamar `f.write()` sem usar a palavra reservada `with` ou chamar `f.close()` **pode** resultar nos argumentos de `f.write()` não serem completamente escritos no disco, mesmo se o programa for encerrado com êxito.

Depois que um arquivo é fechado, seja por uma instrução `with` ou chamando `f.close()`, as tentativas de usar o arquivo falharão automaticamente.

```
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

7.2.1. Métodos de objetos arquivo

Para simplificar, o resto dos exemplos nesta seção assumem que um objeto arquivo chamado `f` já foi criado.

Para ler o conteúdo de um arquivo, chame `f.read(tamanho)`, que lê um punhado de dados devolvendo-os como uma string (em modo texto) ou bytes (em modo binário). *tamanho* é um argumento numérico opcional. Quando *tamanho* é omitido ou negativo, todo o conteúdo do arquivo é lido e devolvido; se o arquivo é duas vezes maior que memória da máquina, o problema é seu. Caso contrário, no máximo *tamanho* caracteres (em modo texto) ou *tamanho* bytes (em modo binário) são lidos e devolvidos. Se o fim do arquivo for atingido, `f.read()` devolve uma string vazia ('').

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

O método `f.readline()` lê uma única linha do arquivo; o caractere de quebra de linha (`\n`) é mantido ao final da string, e só é omitido na última linha do arquivo, se o arquivo não terminar com uma quebra de linha. Isso elimina a ambiguidade do valor retornado; se `f.readline()` retorna uma string vazia, o fim do arquivo foi atingido. Linhas em branco são representadas por um `'\n'` – uma string contendo apenas o caractere terminador de linha.

```
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

Uma maneira alternativa de ler linhas do arquivo é iterar diretamente pelo objeto arquivo. É eficiente, rápido e resulta em código mais simples:

```
>>> for line in f:
...     print(line, end='')
...
This is the first line of the file.
Second line of the file
```

Se desejar ler todas as linhas de um arquivo em uma lista, pode-se usar `list(f)` ou `f.readlines()`.

`f.write(string)` escreve o conteúdo de *string* para o arquivo, retornando o número de caracteres escritos.

```
>>> f.write('This is a test\n')
15
```

Outros tipos de objetos precisam ser convertidos – seja para uma string (em modo texto) ou para bytes (em modo binário) – antes de escrevê-los:

```
>>> value = ('the answer', 42)
>>> s = str(value) # convert the tuple to string
>>> f.write(s)
18
```


`f.tell()` retorna um inteiro dando a posição atual do objeto arquivo, no arquivo representado, como número de bytes desde o início do arquivo, no modo binário, e um número ininteligível, quando no modo de texto.

Para mudar a posição, use `f.seek(offset, de_onde)`. A nova posição é computada pela soma do deslocamento *offset* a um ponto de referência especificado pelo argumento *de_onde*. Se o valor de *de_onde* é 0, a referência é o início do arquivo, 1 refere-se à posição atual, e 2 refere-se ao fim do arquivo. Este argumento pode ser omitido e o valor padrão é 0, usando o início do arquivo como referência.

```
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)          # Go to the 6th byte in the file
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2)      # Go to the 3rd byte before the end
13
>>> f.read(1)
b'd'
```

Em arquivos texto (abertos sem um `b`, em modo string), somente *seeks* relativos ao início do arquivo serão permitidos (exceto se for indicado o final do arquivo, com `seek(0, 2)`) e o único valor válido para *offset* são aqueles retornados por chamada à `f.tell()`, ou zero. Qualquer outro valor para *offset* produz um comportamento indefinido.

Objetos arquivo tem alguns métodos adicionais, como `isatty()` e `truncate()` que não são usados com frequência; consulte a Biblioteca de Referência para um guia completo de objetos arquivo.

ATIVIDADES

1. A ACME Inc., uma empresa de 500 funcionários, está tendo problemas de espaço em disco no seu servidor de arquivos. Para tentar resolver este problema, o Administrador de Rede precisa saber qual o espaço ocupado pelos usuários, e identificar os usuários com maior espaço ocupado. Através de um programa, baixado da Internet, ele conseguiu gerar o seguinte arquivo, chamado "usuarios.txt":

```
alexandre 456123789
anderson 1245698456
antonio 123456456
carlos 91257581
cesar 987458
rosemary 789456125
```

Neste arquivo, o nome do usuário possui 15 caracteres. A partir deste arquivo, você deve criar um programa que gere um relatório, chamado "relatório.txt", no seguinte formato:

```
ACME Inc.                               Uso do espaço em disco pelos usuários
-----
Nr.  Usuário        Espaço utilizado     % do uso

1    alexandre      434,99 MB           16,85%
2    anderson       1187,99 MB          46,02%
3    antonio        117,73 MB           4,56%
4    carlos         87,03 MB            3,37%
5    cesar          0,94 MB             0,04%
6    rosemary       752,88 MB           29,16%

Espaço total ocupado: 2581,57 MB
Espaço médio ocupado: 430,26 MB
```

O arquivo de entrada deve ser lido uma única vez, e os dados armazenados em memória, caso sejam necessários, de forma a agilizar a execução do programa. A conversão da espaço ocupado em disco, de bytes para megabytes deverá ser feita através de uma função separada, que será chamada pelo programa principal. O cálculo do percentual de uso também deverá ser feito através de uma função, que será chamada pelo programa principal.