1.9. Entrada e Saída

Geralmente, precisamos interagir com os usuários para obter dados ou fornecer algum tipo de resultado. A maioria dos programas hoje usa uma caixa de diálogo como forma de pedir ao usuário para fornecer algum tipo de entrada. Embora o Python tenha uma maneira de criar caixas de diálogo, há uma função muito mais simples que podemos usar. O Python nos fornece uma função que permite pedir a um usuário para inserir alguns dados e retorna um referência aos dados na forma de uma string. A função é chamada `input` .

A função `input` do Python recebe um único parâmetro que é uma string. Essa string é frequentemente chamada de **prompt** porque contém um texto informativo que informa o usuário para digitar alguma coisa. Por exemplo, você pode chamar a função `input` da seguinte forma:

```
umNome = input('Por favor digite o seu nome: ')
```

Agora, o que o usuário digitar após o prompt será armazenado na variável `umNome` . Usando a função `input` , podemos facilmente escrever instruções que solicitarão ao usuário que insira dados e, em seguida, incorporar esses dados no processamento. Por exemplo, nas duas seguintes instruções, a primeira pergunta ao usuário pelo nome e a segunda imprime o resultado de algum processamento simples baseado na string que fornecida.

Run Load History Show CodeLens

```
1 umNome = input('Por favor digite o seu nome: ')
2 print("Seu nome todo em maiúsculas é", umNome.upper(),
3       "e tem comprimento", len(umNome))
4
```

ActiveCode: 1 A Função ``input`` Retorna Uma String (strstuff)

É importante notar que o valor retornado pela função `input` será uma string representando os caracteres exatos que foram inserido após o prompt. Se você quiser que essa string seja interpretada como outro tipo, você deve fornecer a conversão de tipo explicitamente. Nas declarações abaixo, a string inserida pelo usuário é convertida em um float que pode ser usado em seguida para processamento aritmético.

```
raio_str = input("Por favor digite o raio do círculo: ")
raio = float(raio_str)
diametro = 2 * raio
```

1.9.1. Formatação de Strings

andoDados.html)

Nós já vimos que a função `print` fornece uma maneira muito simples para exibir valores de saída de um programa em Python. O `print` aceita zero ou mais parâmetros e os exibe usando um único espaço em branco como o separador padrão. É possível alterar o caracter separador, definindo o parâmetro opcional

`sep` . Além disso, cada `print` termina com um caractere de nova linha por default. Esse comportamento pode ser alterado, definindo o parâmetro opcional `end` . Estas variações são mostradas a seguir:

```
>>> print("Olá")
Olá
>>> print("Olá","Mundo")
Olá Mundo
>>> print("Olá","Mundo", sep="***")
Olá***Mundo
>>> print("Olá","Mundo", end="***")
Olá Mundo***
>>>
```

Geralmente, é útil ter mais controle sobre a aparência da sua saída. Felizmente, o Python nos fornece uma alternativa chamada **string de formatação**. Uma string de formatação é um modelo (*template*) no qual palavras ou espaços que permanecerão constantes são combinados com espaços reservados para variáveis que serão inseridas na string. Por exemplo, o comando

```
print(umNome, "tem", anos, "anos de idade.")
```

contém as palavras `tem `` e ``anos de idade` que são constantes, mas o nome e a idade dependem do valor das variáveis no momento da execução. Usando uma string de formatação, nós escrevemos o comando anterior como

```
print("%s tem %d anos de idade." % (umNome, anos))
```

Este exemplo simples ilustra uma nova expressão com strings. O operador `%` é chamado **operador de formatação**. O lado esquerdo da expressão contém o modelo ou string de formatação, e o lado direito contém uma coleção de valores que serão substituídos na string de formatação. Observe que o número de valores na coleção do lado direito corresponde ao número de caracteres `%` na string de formatação. Os valores são usados na sequência, da esquerda para a direita da coleção e inseridos na string de formatação.

Vamos ver os dois lados dessa expressão de formatação em mais detalhes. A string de formatação pode conter uma ou mais especificações de conversão. Um caractere de conversão informa ao operador de formatação que tipo de dado vai ser inserido nessa posição na string. No exemplo acima, o `%s` especifica uma string, enquanto o `%d` especifica um inteiro. Outras especificações possíveis incluem `i` , `u` , `f` , `e` , `g` , `c` ou `%` . A Tabela 9 resume todos os diversos tipos de caracteres de conversão.

Tabela 9: Conversão de Caracteres para Formatação de Strings

Caractere de conversão	Usado para
`d` , `i`	Inteiro (int)
`u`	Inteiro sem sinal
`f`	Ponto flutuante (float) como m.ddddd

andoDados.html)

Caractere de conversão	Usado para
e	Ponto flutuante (float) como m.ddddde+/-xx
E	Ponto flutuante (float) como m.dddddE+/-xx
g	Usa %e com expoentes menores que -4 ou maiores que $+5$, caso contrário usa %f
c	Inserir um único caractere
s	String, ou qualquer tipo do Python que possa ser convertido para string usando a função str
%	Inserir um caractere %

Além do caractere de conversão, você também pode incluir um modificador de formato entre o caractere % e o caractere de conversão. Modificadores de formato podem ser usados para justificar o valor à esquerda ou à direita, dentro de um campo com determinada largura. Modificadores também podem ser usados para especificar a largura do campo bem como um número de dígitos após o ponto decimal. A Tabela 10 explica esses modificadores de formato

Tabela 10: Opções adicionais de formatação

Modificador	Exemplo	Descrição
número	%20d	Coloca o valor em um campo de 20 caracteres
-	%-20d	Coloca o valor em um campo de 20 caracteres, justificado à esquerda
+	%+20d	Coloca o valor em um campo de 20 caracteres, justificado à direita
0	%020d	Coloca o valor em um campo de 20 caracteres, preenchido com zeros à esquerda
.	%20.2f	Coloca o valor em um campo de 20 caracteres, com 2 caracteres à direita do ponto decimal
(nome)	%(nome)d	Coloca o valor associado à chave nome de um dicionário fornecido

O lado direito do operador de formatação é uma coleção de valores que será inserido na string de formatação. A coleção deve ser uma tupla ou um dicionário. Se a coleção for uma tupla, os valores são inseridos na mesma ordem da tupla. Ou seja, o primeiro elemento na tupla corresponde ao primeiro caractere de conversão na string de formatação. Se a coleção for um dicionário, os valores são inseridos de acordo com sua chaves. Neste caso, todos os caracteres de formato devem usar o modificador (nome) para especificar o nome da chave.

andoDados.html)

```
>>> preço = 24
>>> item = "banana"
>>> print("O preço do kilo de %s é %d reais"%(item,preço))
O preço do kilo de banana é 24 reais
>>> print("O preço do kilo de %+10s é %5.2d reais"%(item,preço))
O preço do kilo de     banana é 24.00 reais
>>> print("O preço do kilo de %+10s é %10.2f reais"%(item,preço))
O preço do kilo de     banana é      24.00 reais
>>> dicio = {"item":"banana","preço":24}
>>> print("O preço do kilo de %(item)s é %(preço)7.1f reais"%dicio)
O preço do kilo de banana é    24.0 reais
>>>
```

Além das strings de formatação que usam caracteres de conversão e modificadores de formato, as strings do Python também incluem um método `format` que pode ser usado em conjunto com uma nova classe `Formatter` para implementar formatações mais complexas. Mais sobre esses recursos podem ser encontrados no manual de referência do Python.

(08-pegandoDados.html)

(ControlStructures.html)

 | Back to top

andoDados.html)