Mais sobre inteiros, floats e strings

Aula 9

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação

Conteúdo da aula

Introdução

- a linguagem Python suporta fundamentalmente dois tipos de dados numéricos: inteiros e ponto flutuante
- além desses, já vimos cadeias de caracteres, também chamadas *strings*
- veremos os especificadores desses tipos e conversões entre valores de tipos distintos
- o conteúdo desta aula é baseado no **Python 3.x**

Introdução

- a linguagem Python suporta fundamentalmente dois tipos de dados numéricos: inteiros e ponto flutuante
- além desses, já vimos cadeias de caracteres, também chamadas *strings*
- veremos os especificadores desses tipos e conversões entre valores de tipos distintos
- o conteúdo desta aula é baseado no **Python 3.x**

Introdução

- a linguagem Python suporta fundamentalmente dois tipos de dados numéricos: inteiros e ponto flutuante
- além desses, já vimos cadeias de caracteres, também chamadas *strings*
- veremos os especificadores desses tipos e conversões entre valores de tipos distintos
- o conteúdo desta aula é baseado no **Python 3.x**

Introdução

- a linguagem Python suporta fundamentalmente dois tipos de dados numéricos: inteiros e ponto flutuante
- além desses, já vimos cadeias de caracteres, também chamadas *strings*
- veremos os especificadores desses tipos e conversões entre valores de tipos distintos
- o conteúdo desta aula é baseado no **Python 3.x**

Tipos inteiros

- forma geral de um especificador de conversão na da função `print`:

```
%[flags][comprimento][.precisão]conversor
```

- onde as *flags* podem ser:

Flag	Significado
–	Valor alinhado à esquerda
+	Valor precedido por + ou –
espaço	Valor positivo precedido por espaço
0	Número preenchido com zeros à esquerda

- podemos utilizar os conversores `d`, `i`, `o`, `x` ou `X`.

Tipos inteiros

- forma geral de um especificador de conversão na da função `print`:

```
%[flags][comprimento][.precisão]conversor
```

- onde as *flags* podem ser:

Flag	Significado
–	Valor alinhado à esquerda
+	Valor precedido por + ou –
espaço	Valor positivo precedido por espaço
0	Número preenchido com zeros à esquerda

- podemos utilizar os conversores `d`, `i`, `o`, `x` ou `X`.

Tipos inteiros

- forma geral de um especificador de conversão na da função `print`:

```
%[flags][comprimento][.precisão]conversor
```

- onde as *flags* podem ser:

Flag	Significado
–	Valor alinhado à esquerda
+	Valor precedido por + ou –
espaço	Valor positivo precedido por espaço
0	Número preenchido com zeros à esquerda

- podemos utilizar os conversores `d`, `i`, `o`, `x` ou `X`.

Tipos inteiros

Especificador	Significado
`d`	inteiro em notação decimal
`i`	inteiro (em notação decimal)
`o`	inteiro não sinalizado em notação octal
`x` ou `X`	inteiro não sinalizado em notação hexadecimal

Números com ponto flutuante

- forma geral do especificador de conversão na função `print`:

```
%[flags][comprimento][.precisão]conversor
```

- `%e`: mostra um número com ponto flutuante no formato exponencial ou notação científica
- `%f`: mostra um número com ponto flutuante no formato com casas decimais fixas, sem expoente
- `%g`: mostra o número com ponto flutuante no formato exponencial ou com casas decimais fixas, dependendo de seu tamanho

Números com ponto flutuante

- forma geral do especificador de conversão na função `print`:

```
%[flags][comprimento][.precisão]conversor
```

- `%e`: mostra um número com ponto flutuante no formato exponencial ou notação científica
- `%f`: mostra um número com ponto flutuante no formato com casas decimais fixas, sem expoente
- `%g`: mostra o número com ponto flutuante no formato exponencial ou com casas decimais fixas, dependendo de seu tamanho

Números com ponto flutuante

- forma geral do especificador de conversão na função `print`:

```
%[flags][comprimento][.precisão]conversor
```

- `%e`: mostra um número com ponto flutuante no formato exponencial ou notação científica
- `%f`: mostra um número com ponto flutuante no formato com casas decimais fixas, sem expoente
- `%g`: mostra o número com ponto flutuante no formato exponencial ou com casas decimais fixas, dependendo de seu tamanho

Strings

- Podemos utilizar operadores aritméticos com strings (contudo, o significado é diferente do que acontece com números)
- Ao digitar no console do Python `help(str)` são exibidas as várias funções que estão disponíveis para variáveis que estejam armazenando strings
- A função `len` funciona para vários tipos, não apenas para strings, por isso ela **não** é uma função específica do tipo e por esse motivo utilizamos `len(minha_string)`

Strings

- Podemos utilizar operadores aritméticos com strings (contudo, o significado é diferente do que acontece com números)
- Ao digitar no console do Python `help(str)` são exibidas as várias funções que estão disponíveis para variáveis que estejam armazenando strings
- A função `len` funciona para vários tipos, não apenas para strings, por isso ela **não** é uma função específica do tipo e por esse motivo utilizamos `len(minha_string)`

Strings

- Podemos utilizar operadores aritméticos com strings (contudo, o significado é diferente do que acontece com números)
- Ao digitar no console do Python `help(str)` são exibidas as várias funções que estão disponíveis para variáveis que estejam armazenando strings
- A função `len` funciona para vários tipos, não apenas para strings, por isso ela **não** é uma função específica do tipo e por esse motivo utilizamos `len(minha_string)`

Strings

- forma geral do especificador de conversão na função `print`:

```
%[flags][comprimento]s
```

Conversão de tipos

- na linguagem Python, podemos realizar **conversões explícitas**
- utilizamos **operadores de conversão de tipo**, ou ***casts***:

```
(nome-do-tipo) expressão
```

- o tipo pode ser, por exemplo, **int**, **float** ou **str**

Conversão de tipos

- na linguagem Python, podemos realizar **conversões explícitas**
- utilizamos **operadores de conversão de tipo**, ou ***casts***:

```
(nome-do-tipo) expressão
```

- o tipo pode ser, por exemplo, **int**, **float** ou **str**

Exercícios

1. Dada uma tripla composta por um símbolo de operação aritmética $(+, -, *$ ou $/)$ e dois números reais (p. ex. `2 * 3`), calcule o resultado ao efetuar a operação indicada para os dois números.

Exercícios

2. Um matemático italiano da idade média conseguiu modelar o ritmo de crescimento da população de coelhos através de uma sequência de números naturais que passou a ser conhecida como **sequência de Fibonacci**. A sequência de Fibonacci é dada pela seguinte sequência de números:

   $$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

   Observe que um elemento arbitrário da sequência é resultante da soma dos dois elementos imediatamente anteriores. Dessa forma, a sequência de Fibonacci também pode ser descrita pela seguinte fórmula de recorrência:

   $$\begin{cases} F_1 &= 1\,, \\ F_2 &= 1\,, \\ F_i &= F_{i-1} + F_{i-2}\,, \quad \textit{para } i \geqslant 3\,. \end{cases}$$

   Escreva uma função `def Fib(n):` que dado $n \geqslant 1$ devolva F_n.

3. Os babilônios descreveram a mais de 4 mil anos um método para calcular a raiz quadrada de um número. Esse método ficou posteriormente conhecido como método de Newton. Dado um número x, o método parte de um chute inicial y para o valor da raiz quadrada de x e sucessivamente encontra aproximações desse valor calculando a média aritmética de y e de x/y. O exemplo a seguir mostra o método em funcionamento para o cálculo da raiz quadrada de 3, com chute inicial 1:

x	y	x/y	$(y+x/y)/2$
3	1	3	2
3	2	1.5	1.75
3	1.75	1.714286	1.732143
3	1.732143	1.731959	1.732051
3	1.732051	1.732051	1.732051

3. (*continuação*)
   (a) Escreva uma função com a seguinte interface:

   ```
   def raiz(x, ε):
   ```

   que receba um número real positivo x e um número real ε e calcule a raiz quadrada de x usando o método de Newton, até que o valor absoluto da diferença entre dois valores consecutivos de y seja menor que ε. Sua função deve **devolver** a raiz calculada **E** a quantidade de passos realizados para obtenção da raiz de x.

   (b) Usando a função do item anterior, escreva um programa que, dados x e ε, escreva na tela a raiz de x com precisão ε e a quantidade de passos para calcular a raiz pelo método de Newton. Exemplo:

   Se $n = 5$ e $\varepsilon = 0.018$, os valores impressos serão 1.732143 e 3.

Exercícios

4. Dizemos que um número natural é **triangular** se é produto de três números naturais consecutivos.
   Exemplo:
   120 é triangular, pois $4 \cdot 5 \cdot 6 = 120$.
   (a) Escreva uma função com a seguinte interface:
   ```
   def triangular(n):
   ```
   que receba um número natural n, devolva *True* se n é triangular e *False* caso contrário.
   (b) Usando a função do item anterior, escreva um programa que, dados um número natural n, informe se ele é triangular.
5. Dados dois números inteiros positivos a e b, representando a fração a/b, escreva um programa que reduz a/b para uma fração irredutível.
   Exemplo:
   Se a entrada é $9/12$ a saída tem de ser $3/4$.

Exercícios

6. (a) Escreva uma função com a seguinte interface:

```
def primo(p):
```

que receba um número inteiro positivo p e verifique se p é primo, devolvendo *True* em caso positivo e *False* em caso negativo.

(b) Usando a função do item anterior, escreva um programa que, dados n números inteiros positivos, calcular a soma dos que são primos.

Exemplo:

Se $n = 5$ e a sequência é $13, 9, 14, 7, 73$, então a saída deverá ser 93, já que $13, 7$ e 73 são números primos.

Exercícios

7. Um número inteiro positivo pode ser **decomposto** como um produto de dois ou mais números. Neste caso, esses números são chamados de **fatores** da decomposição. Por exemplo, $18 = 3 \times 6$ é uma decomposição do número 18 nos dois fatores 3 e 6. Quando os fatores do produto da decomposição de um número inteiro positivo n são todos números primos, dizemos que tal decomposição é uma **fatoração** ou **decomposição em fatores primos**. Por exemplo, $18 = 2 \times 3 \times 3$ é uma fatoração do número 18. Uma forma ilustrativa de fazer a fatoração de um número é mostrada no exemplo abaixo, onde ocorre a fatoração, ou decomposição em fatores primos, do número 180:

Exercícios

7. (*continuação*)

180	2
90	2
45	3
15	3
5	5
1	$2^2 \cdot 3^2 \cdot 5^1 = 180$

Dado um número inteiro positivo, determine sua fatoração, calculando também a multiplicidade de cada fator.
Exemplo:
  Se $n = 600$ a saída deve ser
  fator 2 multiplicidade 3
  fator 3 multiplicidade 1
  fator 5 multiplicidade 2

Exercícios

8. Uma maneira de calcular o valor do número π é utilizar a seguinte série:

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

Escreva um programa que calcule e imprima o valor de π através da série acima, com precisão de 4 casas decimais. Para obter a precisão desejada, adicionar apenas os termos cujo valor absoluto seja maior ou igual a 0.0001.

Exercícios

9. Escreva um programa que traduz um número de telefone alfabético de 8 dígitos em um número de telefone na forma numérica. Suponha que a entrada é sempre dada em caracteres maiúsculos.
   Exemplo:
   Se a entrada é `URGENCIA` a saída deve ser `87436242`. Se a entrada é `1111FOGO` a saída deve ser `11113646`.
   Se você não possui um telefone, então as letras que correspondem às teclas são as seguintes: 2=ABC, 3=DEF, 4=GHI, 5=JKL, 6=MNO, 7=PQRS, 8=TUV e 9=WXYZ.
10. Escreva um programa que receba dois números inteiros a e b e uma string *op* (contendo um único caractere), tal que *op* pode ser um dos operadores aritméticos disponíveis na linguagem Python: (`+`, `-`, `*`, `**`, `/`, `//`, `%`), realize a operação a *op* b e mostre o resultado na saída.

Exercícios

11. Um número inteiro positivo é chamado de **palíndromo** se lido da esquerda para a direita e da direita para a esquerda representa o mesmo número.
    A operação *inverta e adicione* é curiosa e divertida. Iniciamos com um número inteiro positivo, invertemos seus dígitos e adicionamos o número invertido ao original. Se o resultado da adição não é um palíndromo, repetimos esse processo até obtermos um número palíndromo.
    Por exemplo, se começamos com o número 195, obtemos o número 9339 como o palíndromo resultante desse processo após 4 adições:

$$\begin{array}{r} 195 \\ +\ 591 \\ \hline 786 \end{array} \qquad \begin{array}{r} 786 \\ +\ 687 \\ \hline 1473 \end{array} \qquad \begin{array}{r} 1473 \\ +\ 3741 \\ \hline 5214 \end{array} \qquad \begin{array}{r} 5214 \\ +\ 4125 \\ \hline 9339 \end{array}$$

Exercícios

11. (continuação)
    Este processo leva a palíndromos em poucos passos para quase todos os números inteiros positivos. Mas existem exceções interessantes. O número 196 é o primeiro número para o qual nenhum palíndromo foi encontrado por este processo. Entretanto, nunca foi provado que tal palíndromo não existe.
    (a) Escreva uma função com a seguinte interface:

    ```
    def inverte(n):
    ```

    que receba um número inteiro positivo n e devolva esse número invertido.

Exercícios

11. (continuação)
    (b) Escreva um programa que receba um número inteiro $k > 0$ que representa a quantidade de casos de teste. Para cada caso de teste, receba um número inteiro $n > 0$ e compute um número palíndromo p através da operação *inverta a adicione* descrita acima (use a função do item (a) e some o número invertido ao original), mostrando na saída o número de passos para encontrar o palíndromo e também o próprio palíndromo. Caso um número inteiro de entrada use mais que 1.000 passos ou ultrapasse o valor 1×10^9, escreva `? ?` na saída.

Exercícios

12. O **mínimo múltiplo comum** entre dois números inteiros positivos pode ser calculado conforme o esquema abaixo ilustrado:

8	36	2
4	18	2
2	9	2
1	9	3
1	3	3
1	1	$2^3 \cdot 3^2 = 72$

   (a) Escreva uma função com a seguinte interface:

```
def divisao(m, n, d):
```

   que receba três números inteiros positivos m, n e d e devolva dois valores. O primeiro valor devolvido deve ser o quociente da divisão de m por d caso m seja divisível por d, ou o próprio valor m caso contrário. O segundo valor devolvido segue a mesma lógica do primeiro, mas utilizando n ao invés de m.

Exercícios

12. (*continuação*)

    (b) Escreva um programa que receba um número inteiro $k > 0$ e uma sequência de k pares de números inteiros positivos m e n e calcule, usando a função do item (a), o mínimo múltiplo comum entre m e n.