Funções
Aula 6

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação

Conteúdo da aula

Motivação

- uma grande tarefa pode ser solucionada através de sua divisão em sub-tarefas e da combinação de suas soluções parciais
- em programação essas soluções menores são chamadas de módulos de programação e fazem parte de todo processo de construção de programas para solução de problemas reais
- módulos são uma ferramenta da programação estruturada que fornecem ao(à) programador(a) um mecanismo para construir programas que são fáceis de escrever, ler, compreender, corrigir, modificar e manter

Motivação

- uma grande tarefa pode ser solucionada através de sua divisão em sub-tarefas e da combinação de suas soluções parciais
- em programação essas soluções menores são chamadas de módulos de programação e fazem parte de todo processo de construção de programas para solução de problemas reais
- módulos são uma ferramenta da programação estruturada que fornecem ao(à) programador(a) um mecanismo para construir programas que são fáceis de escrever, ler, compreender, corrigir, modificar e manter

Motivação

- uma grande tarefa pode ser solucionada através de sua divisão em sub-tarefas e da combinação de suas soluções parciais
- em programação essas soluções menores são chamadas de módulos de programação e fazem parte de todo processo de construção de programas para solução de problemas reais
- módulos são uma ferramenta da programação estruturada que fornecem ao(à) programador(a) um mecanismo para construir programas que são fáceis de escrever, ler, compreender, corrigir, modificar e manter

Motivação

- na linguagem Python, essa característica em um código é obtida através do uso de funções
- cada função na linguagem Python pode ser vista como um pequeno programa com suas próprias declarações de variáveis e sentenças de programação
- além de facilitar a compreensão e a modificação de um programa e evitar a duplicação de código usado mais que uma vez, as funções podem ser usadas não só no programa para o qual foram projetadas, mas também em outros programas
- já temos utilizado funções há bastante tempo, agora veremos como escrever nossas próprias funções

Motivação

- na linguagem Python, essa característica em um código é obtida através do uso de funções
- cada função na linguagem Python pode ser vista como um pequeno programa com suas próprias declarações de variáveis e sentenças de programação
- além de facilitar a compreensão e a modificação de um programa e evitar a duplicação de código usado mais que uma vez, as funções podem ser usadas não só no programa para o qual foram projetadas, mas também em outros programas
- já temos utilizado funções há bastante tempo, agora veremos como escrever nossas próprias funções

Motivação

- na linguagem Python, essa característica em um código é obtida através do uso de funções
- cada função na linguagem Python pode ser vista como um pequeno programa com suas próprias declarações de variáveis e sentenças de programação
- além de facilitar a compreensão e a modificação de um programa e evitar a duplicação de código usado mais que uma vez, as funções podem ser usadas não só no programa para o qual foram projetadas, mas também em outros programas
- já temos utilizado funções há bastante tempo, agora veremos como escrever nossas próprias funções

Motivação

- na linguagem Python, essa característica em um código é obtida através do uso de funções
- cada função na linguagem Python pode ser vista como um pequeno programa com suas próprias declarações de variáveis e sentenças de programação
- além de facilitar a compreensão e a modificação de um programa e evitar a duplicação de código usado mais que uma vez, as funções podem ser usadas não só no programa para o qual foram projetadas, mas também em outros programas
- já temos utilizado funções há bastante tempo, agora veremos como escrever nossas próprias funções

Noções iniciais

```
# Recebe dois números reais e devolve a média aritmética deles
def media(a, b):
    return (a + b) / 2


x = int(input())
y = int(input())
m = media(x, y)
print(m)

# outra opção
print("Média: %g" % media(x, y))
```

A indentação define o trecho de código que compõe a função!

Noções iniciais

```
# Recebe dois números reais e devolve a média aritmética deles
def media(a, b):
    return (a + b) / 2


x = int(input())
y = int(input())
m = media(x, y)
print(m)

# outra opção
print("Média: %g" % media(x, y))
```

A indentação define o trecho de código que compõe a função!

Noções iniciais

```
# Recebe dois números reais e devolve a média aritmética deles
def media(a, b):
    return (a + b) / 2


x = int(input())
y = int(input())
m = media(x, y)
print(m)

# outra opção
print("Média: %g" % media(x, y))
```

A indentação define o trecho de código que compõe a função!

Noções iniciais

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Recebe dois números reais e devolve a média aritmética deles
def media(a, b):
    return (a + b) / 2

# Recebe três números reais na mesma linha e
# calcula a média aritmética para cada par

x, y, z = input("Informe três valores: ").split()
x, y, z = float(x), float(y), float(z)
print("Média de %g e %g é %g" % (x, y, media(x, y)))
print("Média de %g e %g é %g" % (x, z, media(x, z)))
print("Média de %g e %g é %g" % (y, z, media(y, z)))

exit(0)
```

Definição e chamada de funções

- uma **função** da linguagem Python é um trecho de código que pode receber um ou mais valores de entrada armazenados em variáveis, chamados de **parâmetros (de entrada)**, que realiza algum processamento específico e que pode devolver um ou mais valores de saída
- construímos funções para resolver pequenos problemas bem específicos e, em conjunto com outras funções, resolver problemas maiores e mais complexos

Definição e chamada de funções

- uma **função** da linguagem Python é um trecho de código que pode receber um ou mais valores de entrada armazenados em variáveis, chamados de **parâmetros (de entrada)**, que realiza algum processamento específico e que pode devolver um ou mais valores de saída
- construímos funções para resolver pequenos problemas bem específicos e, em conjunto com outras funções, resolver problemas maiores e mais complexos

Definição e chamada de funções

A definição de uma função na linguagem Python de modo geral fornece duas informações importantes:

- seu identificador
- seus parâmetros de entrada

A quantidade de valores devolvidos pode ser deduzida olhando a quantidade de valores ou variáveis após o `return`

Definição e chamada de funções

Toda função deve necessariamente possuir:

- uma **interface**, que faz a comunicação entre a função e o meio exterior; a interface é definida pela primeira linha da função, onde especificamos seu identificador e a lista de parâmetros de entrada separados por vírgulas e envolvidos por um par de parênteses; e
- um **corpo**, que realiza o processamento sobre os valores armazenados nos parâmetros de entrada, e também nas variáveis utilizadas internamente na função, e devolve os valores de saída; o corpo de uma função é composto pela sua lista de comandos.

Definição e chamada de funções

Forma geral de uma **definição de uma função**:

```
def identificador(parâmetros):
    sentença_1
    .
    .
    .
    sentença_n
```

Definição e chamada de funções

- uma função não consegue acessar variáveis de fora dela, para isso utilizamos parâmetros
- quando passamos como parâmetros da uma função alguma variável contendo tipos básicos (inteiro, float, string, ...), alterar o parâmetro não implica em alteração do valor armazenado pela variável no corpo principal do programa, mesmo que o nome seja o mesmo:

```
def funcao_teste(a):
    a = a * 2
    print(a)

a = 8
print(a)
funcao_teste(a)
print(a)
```

- as variáveis criadas no corpo de uma função pertencem exclusivamente àquela função e não podem ser examinadas fora da função, para isso devolvemos valores com o `return`

Definição e chamada de funções

- uma função não consegue acessar variáveis de fora dela, para isso utilizamos parâmetros
- quando passamos como parâmetros da uma função alguma variável contendo tipos básicos (inteiro, float, string, ...), alterar o parâmetro não implica em alteração do valor armazenado pela variável no corpo principal do programa, mesmo que o nome seja o mesmo:

```
def funcao_teste(a):
    a = a * 2
    print(a)

a = 8
print(a)
funcao_teste(a)
print(a)
```

- as variáveis criadas no corpo de uma função pertencem exclusivamente àquela função e não podem ser examinadas fora da função, para isso devolvemos valores com o `return`

Definição e chamada de funções

- uma função não consegue acessar variáveis de fora dela, para isso utilizamos parâmetros
- quando passamos como parâmetros da uma função alguma variável contendo tipos básicos (inteiro, float, string, ...), alterar o parâmetro não implica em alteração do valor armazenado pela variável no corpo principal do programa, mesmo que o nome seja o mesmo:

```
def funcao_teste(a):
    a = a * 2
    print(a)

a = 8
print(a)
funcao_teste(a)
print(a)
```

- as variáveis criadas no corpo de uma função pertencem exclusivamente àquela função e não podem ser examinadas fora da função, para isso devolvemos valores com o `return`

Definição e chamada de funções

- a **chamada de função** consiste de um identificador da função seguido por uma lista de argumentos entre parênteses:

```
media(x, y)
imprime_contador(i)
imprime_msg()
```

Definição e chamada de funções

- para uma função que devolva algum valor, esse valor pode ser armazenado em uma variável ou usado em uma expressão aritmética, relacional ou lógica:

```
if media(x, y) > 0:
   print("Média é positiva")
print("A média é %g" % media(x, y))
```

- contudo, se formos utilizar o valor devolvido várias vezes, é mais eficiente armazená-lo em uma variável:

```
m = media(x, y)
if m > 0:
   print("Média é positiva")
print("A média é %g" % m)
```

- o valor devolvido pode ser ignorado, mas isso geralmente não faz sentido:

```
media(x, y)
```

Definição e chamada de funções

- para uma função que devolva algum valor, esse valor pode ser armazenado em uma variável ou usado em uma expressão aritmética, relacional ou lógica:

```
if media(x, y) > 0:
   print("Média é positiva")
print("A média é %g" % media(x, y))
```

- contudo, se formos utilizar o valor devolvido várias vezes, é mais eficiente armazená-lo em uma variável:

```
m = media(x, y)
if m > 0:
   print("Média é positiva")
print("A média é %g" % m)
```

- o valor devolvido pode ser ignorado, mas isso geralmente não faz sentido:

```
media(x, y)
```

Definição e chamada de funções

- para uma função que devolva algum valor, esse valor pode ser armazenado em uma variável ou usado em uma expressão aritmética, relacional ou lógica:

```
if media(x, y) > 0:
   print("Média é positiva")
print("A média é %g" % media(x, y))
```

- contudo, se formos utilizar o valor devolvido várias vezes, é mais eficiente armazená-lo em uma variável:

```
m = media(x, y)
if m > 0:
   print("Média é positiva")
print("A média é %g" % m)
```

- o valor devolvido pode ser ignorado, mas isso geralmente não faz sentido:

```
media(x, y)
```

Definição e chamada de funções

- para devolver um valor em uma função, utilizamos a sentença **`return`** para especificar qual valor será devolvido, com a seguinte forma geral:

```
return expressão
```

- a *`expressão`* em geral é uma constante ou uma variável:

```
return 0

    ou

return estado
```

- expressões mais complexas são possíveis:

```
return (a + b) / 2

    ou

return a * a * a + 2 * a * a - 3 * a - 1
```

Definição e chamada de funções

- para devolver um valor em uma função, utilizamos a sentença **`return`** para especificar qual valor será devolvido, com a seguinte forma geral:

```
return expressão
```

- a ***expressão*** em geral é uma constante ou uma variável:

```
return 0

    ou

return estado
```

- expressões mais complexas são possíveis:

```
return (a + b) / 2

    ou

return a * a * a + 2 * a * a - 3 * a - 1
```

Definição e chamada de funções

- para devolver um valor em uma função, utilizamos a sentença `return` para especificar qual valor será devolvido, com a seguinte forma geral:

```
return expressão
```

- a *expressão* em geral é uma constante ou uma variável:

```
return 0

    ou

return estado
```

- expressões mais complexas são possíveis:

```
return (a + b) / 2

    ou

return a * a * a + 2 * a * a - 3 * a - 1
```

Devolvendo mais de um valor

- na linguagem Python, uma função pode devolver vários valores
- nesse caso, precisamos armazenar os valores devolvidos utilizando atribuição múltipla:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Decompõe um número real, devolvendo a parte
# inteira e a parte fracionária
def decompoe(x):
    parte_int = int(x)
    parte_frac = x - parte_int
    return parte_int, parte_frac

num = float(input())
p_i, p_f = decompoe(num)
print("Número: %f" % num)
print("Parte inteira: %d" % p_i)
print("Parte fracionária: %f" % p_f)

exit(0)
```

Devolvendo mais de um valor

- na linguagem Python, uma função pode devolver vários valores
- nesse caso, precisamos armazenar os valores devolvidos utilizando atribuição múltipla:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Decompõe um número real, devolvendo a parte
# inteira e a parte fracionária
def decompoe(x):
    parte_int = int(x)
    parte_frac = x - parte_int
    return parte_int, parte_frac

num = float(input())
p_i, p_f = decompoe(num)
print("Número: %f" % num)
print("Parte inteira: %d" % p_i)
print("Parte fracionária: %f" % p_f)

exit(0)
```

Exercícios

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

```
def area_triangulo(base, altura):
```

que receba dois números de ponto flutuante que representam a base e a altura de um triângulo e compute e devolva a área desse triângulo.

(b) Escreva um programa que receba um par de números de ponto flutuante que representa a base e a altura de um triângulo, então calcule e escreva a área do triângulo correspondente. Use a função descrita no item (a).

Exercícios

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

```
def maior_de_3(a, b, c):
```

que receba 3 números e devolva o maior valor.

(b) Escreva um programa que receba leia três números e escreva o maior e o menor. Utilize a função acima.

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

```
def idade_em_dias(anos, meses, dias):
```

que receba a idade de uma pessoa em anos, meses e dias e devolva essa idade expressa em dias.

(b) Escreva um programa que receba leia a idade de uma pessoa em anos, meses e dias e escreva essa idade em dias. Utilize a função acima.

Exercícios

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

```
def maior_de_3(a, b, c):
```

que receba 3 números e devolva o maior valor.

(b) Escreva um programa que receba leia três números e escreva o maior e o menor. Utilize a função acima.

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

```
def idade_em_dias(anos, meses, dias):
```

que receba a idade de uma pessoa em anos, meses e dias e devolva essa idade expressa em dias.

(b) Escreva um programa que receba leia a idade de uma pessoa em anos, meses e dias e escreva essa idade em dias. Utilize a função acima.

Exercícios

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

```
def crescente_3(a, b, c):
```

que receba três números inteiros a, b e c, e devolva três números em ordem crescente (o primeiro valor devolvido pelo *return* será o menor, o segundo será o médio, e o terceiro será o maior). Lembre-se que para trocar o conteúdo de duas variáveis, digamos `a` e `b`, podemos fazer `a, b = b, a`, isso pode ajudá-lo a resolver esse problema de forma mais fácil.

(b) Usando a função acima, leia três números e escreva-os em ordem crescente.

Exercícios

5. Em um dado país a moeda corrente possui apenas quatro cédulas de papel: $1, $5, $10 e $20.
   (a) Escreva uma função com a seguinte interface:

```
def cedulas(val):
```

   que receba um número inteiro não-negativo que representa um valor na moeda do país, e então determine e devolva a menor quantidade de cédulas de 1, 5, 10 e 20 necessárias para pagar o valor especificado, nessa ordem. Observe que a função devolverá 4 valores.
   (b) Escreva um programa que receba um número inteiro representando um valor na moeda corrente, e determine a menor quantidade de cédulas para pagar tal valor. Use a função do item (a).

Exercícios

6. Para determinar o número de lâmpadas necessárias para cada aposento de uma residência, existem normas que fornecem o mínimo de potência de iluminação exigida por metro quadrado (m^2) conforme o uso desse ambiente. Suponha que só temos lâmpadas de 60 watts para uso.
   Seja a seguinte tabela de informações sobre possíveis aposentos de uma residência:

Utilização	Classe	Potência/m^2 (W)
quarto	1	15
sala de TV	1	15
salas	2	18
cozinha	2	18
varandas	2	18
escritório	3	20
banheiro	3	20

Exercícios

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

```
def num_lampadas(classe, a, b):
```

   que receba um número inteiro representando a classe de iluminação de um aposento e dois números com ponto flutuante representando suas duas dimensões, e devolva um número inteiro representando o número de lâmpadas necessárias para iluminar adequadamente o aposento.
   (b) Escreva um programa que receba a classe de iluminação de um aposento e as suas dimensões e, usando a função `num_lampadas`, imprima a área do aposento e o número total de lâmpadas necessárias para o aposento