Projeto de funções e instruções condicionais

Marco A L Barbosa

malbarbo.pro.br

Departamento de Informática

Universidade Estadual de Maringá

Conteúdo

Projeto de funções

Introdução

- Vimos que os programas de computadores recebem dados de entrada, fazem processamento e produzem saídas
- Os programas são compostos de funções e cada função recebe parâmetros de entrada, faz o processamento e produz uma saída
- Programar é criar funções!

Introdução

- Vimos anteriormente que podemos criar novas funções através da combinação de funções e operações existentes. A forma inicial para definição de funções é

```
def nome_da_funcao(parametro1, parametro2, ...):
    return expressao
```

- Cada **def** inicia no começo de uma linha e a instrução **return** fica "dentro" do **def**.

- Por enquanto, toda função deve ter exatamente uma instrução **return**.

Receita de projeto de funções

- Vamos seguir uma receita para definir novas funções

    1. Cabeçalho: nome da função e dos parâmetros e `return`
    2. Contrato: tipos dos dados de entrada e saída
    3. Propósito: descrição do que a função faz
    4. Exemplos: saída produzida para algumas entradas
    5. Corpo: o código da função
    6. Teste: verifica se a função se comporta como nos exemplos

Receita de projeto de funções

- Cada etapa depende da anterior, mas as vezes pode ser necessário mudar a ordem
- Por exemplo, talvez você faça primeiro os exemplos para entender melhor o problema e poder escrever o contrato e o propósito
- As vezes você está escrevendo o corpo e encontra uma nova condição e deve voltar e alterar o propósito e os exemplos
- Mas você nunca deve escrever o código diretamente

Exemplo 1

Defina uma função que produza o dobro de um dado valor.

Exemplo 1

Passo 1: cabeçalho - nome da função e dos parâmetros e **`return`**

```
def dobro(x):
    return
```

Exemplo 1

Passo 2: contrato - o que a função consome e produz - tipo dos dados de entrada e saída

```
def dobro(x):
    '''
    Número -> Número
    '''
    return
```

Exemplo 1

Passo 3: propósito - o que a função faz

```
def dobro(x):
    '''
    Número -> Número
    Produz o dobro de x.
    '''
    return
```

Exemplo 1

Passo 4: exemplos - resultado esperado para algumas entradas

```
def dobro(x):
    '''
    Número -> Número
    Produz o dobro de x.
    Exemplos
    >>> dobro(3)     # 2 * 3
    6
    >>> dobro(-2.1)  # 2 * -2.1
    -4.2
    '''
    return
```

Para cada exemplo podemos escrever após o sinal # como o resultado foi obtido.

Esta informação pode ajudar na escrita do corpo da função

Exemplo 1

Passo 5: corpo - o código da função

```
def dobro(x):
    '''
    Número -> Número
    Produz o dobro de x.
    Exemplos
    >>> dobro(3)     # 2 * 3
    6
    >>> dobro(-2.1)  # 2 * -2.1
    -4.2
    '''
    return 2 * x
```

Exemplo 1

Passo 6: teste - a função se comporta como nos exemplos? Testamos na janela de iterações

```
>>> dobro(3)
6
>>> dobro(-2.1)
-4.2
```

Exemplo 1

Se você não estiver usando o Editor Mu (por exemplo, você está usando o QPython3 no telefone), você pode testar a função escrevendo o seguinte no final do arquivo (depois de todos as definições de funções) e executando o programa

```
print(dobro(3))
print(dobro(-2.1))
```

Observe atentamente o que vai aparecer escrito na tela! Cada linha corresponde a um resultado.

Testes automatizados

Testes automatizados

O Python pode verificar automaticamente se todos os exemplos estão corretos

- Chamamos esta verificação de **teste automatizado**
- A linha a seguir deve ser incluída no início do arquivo

```
from doctest import testmod
```

- Para executar o teste automatizado, clique em "Executar" e chame a função `testmod` na janela de interações

```
>>> testmod()
TestResults(failed=0, attempted=2)
```

Testes automatizados

Se você não estiver usando o Editor Mu, você pode executar os testes automatizados escrevendo o seguinte no final do arquivo e executando o programa

```
print(testmod())
```

Testes automatizados

A checagem de um exemplo pode falhar por um de três motivos

- O exemplo está errado
    - Refaça o exemplo para ter certeza que ele está certo
- O programa está errado
    - Neste caso o programador cometeu um erro lógico e o corpo da função deve ser corrigido
- O exemplo e o programa estão errados!

Exemplo 2

Defina uma função que verifique se um número é par.

Exemplo 2

Passo 1: cabeçalho - nome da função e dos parâmetros e **`return`**

```
def par(x):
    return
```

Exemplo 2

Passo 2: contrato - o que a função consome e produz - tipo dos dados de entrada e saída

```
def par(x):
    '''
    Inteiro -> Boolean
    '''
    return
```

Exemplo 2

Passo 3: propósito - o que a função faz

```
def par(x):
    '''
    Inteiro -> Boolean
    Produz True se x é um número par, False caso contrário.
    '''
    return
```

Exemplo 2

Passo 4: exemplos - resultado esperado para algumas entradas

```
def par(x):
    '''
    Inteiro -> Boolean
    Produz True se x é um número par, False caso contrário.
    Exemplos
    >>> par(4) # 4 % 2 == 0
    True
    >>> par(7) # 7 % 2 == 0
    False
    '''
    return
```

Exemplo 2

Passo 5: corpo - o código da função

```
def par(x):
    '''
    Inteiro -> Boolean
    Produz True se x é um número par, False caso contrário.
    Exemplos
    >>> par(4) # 4 % 2 == 0
    True
    >>> par(7) # 7 % 2 == 0
    False
    '''
    return x % 2 == 0
```

Exemplo 2

Passo 6: teste - a função se comporta como nos exemplos? Testamos na janela de iterações

```
>>> testmod()
TestResults(failed=0, attempted=2)
```

Exemplo 3

Defina uma função que converta uma quantidade de segundos para uma quantidade de horas, minutos e segundos equivalente. A quantidade de segundos e de minutos da resposta não podem ser maior que 60.

Exemplo 3

Passo 1: cabeçalho - nome da função e dos parâmetros e **`return`**

```
def hms(segundos):
    return
```

Exemplo 3

Passo 2: contrato - o que a função consome e produz - tipo dos dados de entrada e saída

```
def hms(segundos):
    '''
    Inteiro -> Inteiro, Inteiro, Inteiro
    '''
    return
```

Exemplo 3

Passo 3: propósito - o que a função faz

```
def hms(segundos):
    '''
    Inteiro -> Inteiro, Inteiro, Inteiro
    Converte uma quantidade de segundos em uma quantidade de horas,
    minutos e segundos equivalente, onde o quantidade de minutos
    e segundos não é maior que 60.
    '''
    return
```

Exemplo 3

Passo 4: exemplos - resultado esperado para algumas entradas

```
def hms(segundos):
    '''
    Tipos e descrição omitidos para economizar espaço...
    Exemplos
    >>> # 70 // 3600 equivale a 0 hora e sobra 70 segundos
    >>> # 70 // 60 equivale a 1 minuto e sobra 10 segundos
    >>> hms(70)
    (0, 1, 10)
    >>> # 10798 // 3600 equivale a 2 horas e sobra 3598 segundos
    >>> # 3598 // 60 equivale a 59 minutos e sobra 58 segundos
    >>> hms(10798)
    (2, 59, 58)
    '''
    return
```

Exemplo 3

Passo 5: corpo - o código da função

Observando os comentários dos exemplo podemos observar que

- A quantidade de horas é calculada primeiro fazendo
  ```
  segundos // 3600
  ```
- O cálculo dos minutos e segundos dependem de um resultado intermediário, que é a quantidades de segundos que sobram depois do cálculo da quantidade de horas
  ```
  segundos % 3600 # quantidades de segundos que sobrou
  ```
- A quantidade de minutos é calculada por
  ```
  (segundos % 3600) // 60
  ```
- A quantidade de segundos é calculada por
  ```
  (segundos % 3600) % 60
  ```

Exemplo 3

Então, podemos definir o corpo da função da seguinte forma

```
def hms(segundos):
    return (segundos // 3600, (segundos % 3600) // 60, (segundos % 3600) % 60)
```

Algumas observações sobre o corpo da função

- Não está claro o que cada expressão significa
- A expressão `segundos % 3600` aparece repetida
  - Isto parece não ser relevante pois a expressão é simples, mas e se a expressão fosse complicada?

Variáveis locais

Variáveis locais

- Uma variável local é um nome definido dentro de uma função
- As variáveis locais têm vários propósitos, por enquanto, vamos utilizá-las para armazenar resultados intermediários

Variáveis locais

- Usamos o símbolo =, chamado de **atribuição**, para definir o valor de uma variável local

  A forma inicial da atribuição é

  ```
  nome = expressao
  ```

- Exemplo

  ```
  distancia = velocidade * tempo
  ```

- Observe que do lado esquerdo da atribuição deve aparece o nome da variável local. Uma expressão não é válida do lado esquerdo!

  ```
  velocidade * tempo = distancia  # Inválido!
  ```

Variáveis locais

- Estendemos a forma de criar funções para permitir variáveis locais

```
def nome_da_funcao(parametro1, parametro2, ...):
    variavel1 = expressao1
    variavel2 = expressao2
    ...
    return expressao
```

- Como o Python avalia uma função com variáveis locais?
  - Ele executa as atribuições em sequência, primeiro avalia `expressao1` e atribui o resultado ao nome `variavel1`, em seguida avalia `expressao2` e atribui o resultado ao nome `variavel2`, e assim por diante
  - Por fim ele avalia `expressao` e produz o resultado da função

Variáveis locais

- Vamos considerar a função `hms`

```
def hms(segundos):
    return (segundos // 3600, (segundos % 3600) // 60, (segundos % 3600) % 60)
```

- Podemos podemos reescrevê-la utilizando variáveis locais

```
def hms(segundos):
    h = segundos // 3600
    sobra = segundos % 3600
    m = sobra // 60
    s = sobra % 60
    return (h, m, s)
```

- Neste caso, os resultados ficam mais claros e evitamos digitar a mesma expressão mais do que uma vez. Além disso, a função se parece mais com a forma que calculamos os resultados para os exemplos: usando resultados intermediários.

Variáveis locais

- Assim como um parâmetro, uma variável local só pode ser usada dentro da função e ela deixa de existir quando a função termina

```
def dobro_soma(a, b):
    c = a + b
    return 2 * c

def f(x, y):
    z = dobro_soma(x, y)
    return z + a + b + c

>>> f(2, 3)
?
```

Variáveis locais

- Assim como um parâmetro, uma variável local só pode ser usada dentro da função e ela deixa de existir quando a função termina

```
def dobro_soma(a, b):
    c = a + b
    return 2 * c

def f(x, y):
    z = dobro_soma(x, y)
    return z + a + b + c

>>> f(2, 3)
Erro! As variáveis a, b e c não existe dentro da função f!
```

Variáveis locais

- Uma variável só pode ser utilizada depois que um valor for atribuído para ela

```
def hipotenusa(a, b):
    c = (a2 + b2) ** 0.5
    a2 = a ** 2
    b2 = b ** 2
    return c

>>> hipotenusa(4.0, 5.0)
?
```

Variáveis locais

- Uma variável só pode ser utilizada depois que um valor foi atribuído para ela

```
def hipotenusa(a, b):
    c = (a2 + b2) ** 0.5
    a2 = a ** 2
    b2 = b ** 2
    return c

>>> hipotenusa(4.0, 5.0)
Erro! Variáveis a2 e b2 não definidas!
```

Exemplo 4

Defina uma função que encontre o máximo entre dois valores dados.

Exemplo 4

Passo 1: cabeçalho - nome da função e dos parâmetros e **`return`**

```
def maximo(a, b):
    return
```

Exemplo 4

Passo 2: contrato - o que a função consome e produz - tipo dos dados de entrada e saída

```
def maximo(a, b):
    '''
    Número, Número -> Número
    '''
    return
```

Exemplo 4

Passo 3: propósito - o que a função faz

```
def maximo(a, b):
    '''
    Número, Número -> Número
    Devolve o valor máximo entre a e b.
    '''
    return
```

Exemplo 4

Passo 4: exemplos - resultado esperado para algumas entradas

```
def maximo(a, b):
    '''
    Número, Número -> Número
    Devolve o valor máximo entre a e b.
    Exemplos
    >>> maximo(2, 4) # ????
    4
    >>> maximo(7, 4) # ????
    7
    '''
    return
```

Ops! Até sabemos os resultados esperados dos exemplos, mas como computar estes valores?

Instruções condicionais

Instruções condicionais

- As instruções condicionais permitem que um programa execute instruções diferentes dependendo do valor de uma determinada condição
- A instrução condicional mais comum nas linguagens de programação é o `if`
- Para fazer a função `maximo`, temos que usar a instrução `if`

Instruções condicionais

- A forma preliminar do **`if`** é

```
if condicao:
    consequente
else:
    alternativa
```

(**`if`** é *se* em português e **`else`** é *senão*)

- Como o Python avalia uma instrução **`if`**?
  - Ele primeiro avalia a expressão `condicao`, se o resultado for `True`, então ele executa as instruções no bloco `consequente`, senão (o resultado da expressão `condição` é `False`), ele avalia a instruções no bloco `alternativa`

Instruções condicionais

- A expressão `condicao` pode ser qualquer expressão cujo o resultado seja `True` ou `False`
- Os blocos `consequente` e `alternativa` são compostos de qualquer sequencia de instruções, incluindo atribuições e outras instruções `if`
- Todas as instruções do `consequente` e `alternativa` devem ficar "dentro" do `if` e do `else`, respectivamente
- Por enquanto, cada `if` deve ter um `else` correspondente

Instruções condicionais

- Estendemos a forma de criar funções para permitir instruções condicionais

```
def nome_da_funcao(parametro1, parametro2, ...):
    atribuicao ou instrucao_condicional
    atribuicao ou instrucao_condicional
    ...
    return expressao
```

- Como o Python avalia uma função com atribuições e instruções condicionais?
  - Ele executa as instruções em sequência...
  - Por fim ele avalia `expressao` e produz o resultado da função

Instruções condicionais

- Podemos pensar nos exemplos da função máximo da seguinte forma

```
>>> maximo(7, 4) # 7 > 4 = True, produz 7
7
>>> maximo(2, 4) # 2 > 4 = False, produz 4
4
```

- Pelo comentário do exemplo, podemos ver que o resultado depende da comparação `a > b`. Podemos usar o **`if`** para fazer uma atribuição e utilizar a variável local com a resposta no **`return`**

```
def maximo(a, b):
    if a > b:
        max = a
    else:
        max = b
    return max
```

Exemplo de execução de uma função com `if`

Dado a função:

```
def f(x, y, z):
    if x >= y:
        if x >= z:
            r = x
        else:
            r = z
    else:
        if y >= z:
            r = y
        else:
            r = z
    return r
```

Qual o resultado produzido por:

```
>>> f(4, 5, 2)
?
>>> f(2, 2, 3)
?
```

Exemplo de execução de uma função com `if`

Dado a função:

```
def f(x, y, z):
    if x >= y:
        if x >= z:
            r = x
        else:
            r = z
    else:
        if y >= z:
            r = y
        else:
            r = z
    return m
```

Qual o resultado produzido por:

```
>>> f(4, 5, 2)
5
>>> f(2, 2, 3)
3
```

Instruções condicionais

- A forma geral do `if` é

```
if condicao1:
    consequente1
elif condicao2:
    consequente2
...
elif condicaon:
    consequenten
else:
    alternativa
```

- Como o Python avalia o `if`?
    - Ele avalia cada condição em ordem, quando ele encontra uma condição com valor `True`, ele executa o bloco consequente associado e encerra a execução do `if`. Se nenhum condição for `True` ele executa o bloco `alternativa`

Exemplo 5

Escreva uma função que receba como entrada a cor atual de um semáforo de trânsito e devolva a próxima cor que será ativada (considere um semáforo com três cores: verde, amarelo e vermelho).

O seguinte exemplo pode ajudar no projeto da função

```
>>> proxima_cor_semaforo('verde')
'amarelo'
```

Exemplo 5

```
def proxima_cor_semaforo(cor_atual):
    '''
    String -> String
    Devolve a próxima cor que será mostrada em um semáforo dado a cor_atual.

    Exemplos
    >>> proxima_cor_semaforo('verde')
    'amarelo'
    >>> proxima_cor_semaforo('amarelo')
    'vermelho'
    >>> proxima_cor_semaforo('vermelho')
    'verde'
    '''
    if cor_atual == 'verde':
        cor_prox = 'amarelo'
    elif cor_atual == 'amarelo':
        cor_prox = 'vermelho'
    else:  # podemos assumir que cor_atual == 'vermelho'
        cor_prox = 'verde'
    return cor_prox
```

Exemplo 6

Defina uma função que receba como parâmetros os lados de um triângulo e o classifique em escaleno, isósceles ou equilátero.

Exemplo 6

```
def classifica_triangulo(a, b, c):
    '''
    Número, Número, Número -> String
    Classifica um triângulo de acordo com as medidas a, b e c do seus lados.
    Um triângulo com os três lados iguais é equilátero. Um triângulo com dois
    lados iguais é isósceles e um triângulo com os três lados diferentes é
    escaleno.

    Exemplos
    >>> classifica_triangulo(3, 3, 3)
    'equilátero'
    >>> classifica_triangulo(2, 2, 3)
    'isósceles'
    >>> classifica_triangulo(2, 3, 2)
    'isósceles'
    >>> classifica_triangulo(2, 2, 3)
    'isósceles'
    >>> classifica_triangulo(2, 3, 4)
    'escaleno'
    '''
    return
```

Exemplo 6

```
def classifica_triangulo(a, b, c):
    if a == b and b == c:
        tipo = 'equilátero'
    elif a != b and b != c and a != c:
        tipo = 'escaleno'
    else:
        tipo = 'isósceles'
    return tipo
```

Exemplo 7

Defina uma função que receba como parâmetros os coeficientes de uma equação de segundo grau e determine as suas raízes. Considere as três possibilidades: uma raiz, duas raízes ou nenhum raiz.

Exemplo 7

```
def baskara(a, b, c):
    '''
    Número, Número, Número -> Número ou (Número, Número) ou String
    Calcula as raízes da equação a (x ** 2) + b x + c. Devolve um valor se a
    equação só tem uma raiz, devolve dois valores se a equação tem duas raízes
    ou devolve 'sem raízes' se a equação não tem raiz.

    Exemplos
    >>> baskara(1, 2, -3)
    (1.0, -3.0)
    >>> baskara(1, 4, 4)
    -2.0
    >>> baskara(4, 2, 3)
    'sem raízes'
    '''
    return
```

Exemplo 7

```
def baskara(a, b, c):
    delta = b ** 2 - 4 * a * c
    if delta > 0:
        x1 = (-b + delta ** 0.5) / (2 * a)
        x2 = (-b - delta ** 0.5) / (2 * a)
        r = (x1, x2)
    elif delta == 0:
        r = -b / (2 * a)
    else:
        r = 'sem raízes'
    return r
```

Atividades

Atividades

1. A empresa Feras da Engenharia paga R$ 50,00 por hora para um engenheiro. Cada engenheiro trabalha em média de 20 a 50 horas por semana. Defina uma função que calcule o salário de um engenheiro a partir do número de horas trabalhada.

Atividades

2. Você precisa saber a massa de diversos pequenos tubos de ferro mas está sem uma balança. No entanto, você possui um paquímetro e pode medir com precisão o diâmetro interno e externo e a altura dos tubos. Considerando a densidade do ferro como 7874 kg/m^3, escreva uma função que receba como parâmetro as dimensões de um tubo de ferro e calcule a sua massa.

Atividades

3. Defina uma função que verifique se três medidas podem formar um triângulo. Para formar um triângulo a soma de qualquer duas medidas deve ser maior ou igual do que a terceira medida.

4. O governo do estado deu uma aumento de salário para os funcionários públicos. O percentual de aumento depende do valor do salário atual. Para funcionários que ganham até R$ 1200 o aumento é de 10%, para funcionários que ganham entre R$ 1200 e R$ 3000 o aumento é de 7%, para funcionários que ganham entre R$ 3000 e R$ 8000, o aumento é de 3%, e finalmente, para os funcionários que ganham mais que R$ 8000 não haverá aumento. Defina uma função que calcule o novo salário a partir do salário atual.

Atividades

5. Um número é palíndromo se quando lido da direita para a esquerda ou da esquerda para a direita é idêntico. Ex: 9119, 1221, 5665, 7337. Defina uma função que verifique se um dado número inteiro de 4 dígitos é palíndromo. Considere que o valor de entrada é o próprio número e não os quatro dígitos que compõem o número.

Atividades

Este exercício já fez parte de uma prova!

6. Um construtor precisa calcular a quantidade de azulejos necessários pra azulejar uma determinada parede. Cada azulejo é quadrado e tem 20cm de lado. Ajude o construtor e defina uma função que receba como entrada o comprimento e a altura em metros de uma parede e calcule a quantidade de azulejos inteiros necessários para azulejar a parede. Considere que o construtor nunca perde um azulejo e que recortes de azulejos não são reaproveitados. (Dica: use divisão pelo piso `//`)

Atividades

7. Um determinado investimento paga rendimento anual dependendo do montante aplicado. Para aplicações de até R$ 5.000, o rendimento é de 7%, para aplicações entre R$ 5.000,00 e R$ 10.000,00 é de 8% e para aplicações maiores que R$ 10.000,00 é de 9%. Defina uma função que receba como parâmetro o valor aplicado e calcule o valor do rendimento após 1 ano de investimento.

Atividades

Este exercício já fez parte de uma prova!

8. A nota final em um disciplina é calculada pela média simples de 4 avaliações que valem de 0 a 10. A partir da nota final os alunos ficam em um de três situações: Aprovado, alunos com nota final maior ou igual a 7. Reprovado, alunos com nota menor que 4. Exame, alunos com nota maior igual a 4 e menores que 7. Defina uma função que indique a situação de um aluno dado as 4 notas das suas avaliações.

Atividades

Este exercício já fez parte de uma prova!

9. Cada cidadão de um país, cuja moeda chama dinheiro, tem que pagar imposto sobre a sua renda. Cidadãos que recebem até 1000 dinheiros pagam 5% de imposto. Cidadãos que recebem entre 1000 e 5000 dinheiros pagam 5% de imposto sobre 1000 dinheiros e 10% sobre o que passar de 1000. Cidadãos que recebem mais do 5000 dinheiros pagam 5% de imposto sobre 1000 dinheiros, 10% de imposto sobre 4000 dinheiros e 20% sobre o que passar de 5000. Defina uma função que calcule o imposto que um cidadão deve pagar dado a sua renda.

Atividades

10. Em um determinado jogo os jogadores são classificados em níveis de 0 a 25 e este nível é atualizado semanalmente baseado na quantidade de horas que o jogador jogou o jogo. Os jogadores que jogaram entre 4 e 5 horas permanecem no mesmo nível. Os jogadores que jogaram menos que 4 horas diminuem um nível a cada 1 hora que faltou para alcançar as 4 horas. Os jogadores que jogaram mais que 5 horas aumentam um nível a cada hora jogada além das 5 horas. Desenvolva uma função que recebe o nível atual do jogador e a quantidade de horas jogas em uma semana e calcule o novo nível do jogador.