Introdução à Ciência da Computação

Disciplina: 113913

Prof. Edison Ishikawa

python™

Python 3.0

Aula 5

Funções frutíferas

UnB | CIC

Sumário

- Funções frutíferas
  - Valores de retorno
  - Desenvolvimento de programas
  - Composição
  - Funções *booleanas*
  - Mais recursividade
  - Voto de confiança (*Leap of faith*)
  - Mais um exemplo
  - Checagem de tipos

Valores de retorno

- Já usamos funções nativas do Python que produziram resultados

Exemplo

```
>>> import math
>>> e=math.exp(1.0)
>>>
>>> altura = raio * math.sin(angulo)
```

- Nenhuma das funções que nós escrevemos retornou um valor
- **Funções frutíferas :** funções que retornam valores

Valores de retorno

- Exemplo de função frutífera:

Exemplo: função area

```
>>> import math
>>> def area(raio):
        temp = math.pi * raio ** 2
        return temp
```

- Em uma função frutífera, a instrução **return** inclui um valor de retorno.

Exemplo: função area

```
>>> def area(raio):
        return math.pi * raio ** 2
```

Valores de retorno

- Exemplo de função frutífera:

Exemplo: função area

```
>>> import math
>>> def area(raio):
        temp = math.pi * raio ** 2
        return temp
```

Torna a depuração mais fácil

- Em uma função frutífera, a instrução **return** inclui um valor de retorno.

Exemplo: função area

```
>>> def area(raio):
        return math.pi * raio ** 2
```

Valores de retorno

- Múltiplos comandos **return**: um em cada ramo de uma condicional:

Exemplo:

```
>>>def valorAbsoluto(x):
        if x < 0:
          return -x
        else:
          return x
```

Apenas um desses comandos **return** será exibido. O que NÃO for exibido é um código morto **(dead code)**

Valores de retorno

- É bom assegurar que todo caminho possível dentro do programa encontre uma instrução **return**:

Exemplo:

```
>>>def valorAbsoluto(x):
        if x < 0:
          return -x
        else x > 0:
          return x
```

Valores de retorno

- É bom assegurar que todo caminho possível dentro do programa encontre uma instrução **return**:

Exemplo:

```
>>>def valorAbsoluto(x):
        if x < 0:
          return -x
        else x > 0:
          return x
```

Se **x** for 0, nenhuma condição será verdadeira, e a função terminará sem encontrar um comando **return**

Exemplo:

```
>>> print (valorAbsoluto(0))
None
```

Desenvolvimento de programas

- Até aqui: pequenas funções
- Funções maiores: podem surgir erros em tempo de execução (erros de *runtime)* ou erros semânticos
- Desenvolvimento incremental para lidar com complexidade crescente
- Evitar seções de depuração (*debugging*) muito longas pela adição e teste de somente uma pequena quantidade de código de cada vez

Desenvolvimento de programas

- Exemplo: Encontrar a distância entre dois pontos, dados pelas coordenadas (x1,y1) e (x2,y2)

Teorema de Pitágoras

$$\text{distancia} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Como deveria ser a função *distancia* em Python?
  - Entradas: parâmetros
  - Saída: valor de retorno

Desenvolvimento de programas

Teorema de Pitágoras

distancia = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

- Entradas: (x1, y1) e (x2, y2)
- Saída: distancia (ponto flutuante)

Exemplo:

```
>>> def distancia(x1, y1, x2, y2):
        return 0.0
```

Não está correto

Desenvolvimento de programas

Teorema de Pitágoras

distancia = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

- Testar a função com valores hipotéticos:

Exemplo:

```
>>> def distancia(1, 2, 4, 6):
        return 0.0
```

Desenvolvimento de programas

- Adicionar linhas de código

Exemplo:

```
>>> def distancia(x1, y1, x2, y2):
        dx = x2 - x1
        dy = y2 – y1
        print (“dx vale”, dx)
        print (“dy vale”, dy)
        return 0.0
```

Variáveis temporárias: dx, dy

Código muleta = *scaffolding*

- Testar a função novamente

Desenvolvimento de programas

- Adicionar linhas de código

Exemplo:

```
def distancia(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 – y1
    dquadrado = dx**2 + dy**2
    print (“dquadrado vale: ”, dquadrado)
    return 0.0
```

- Testar a função novamente

Desenvolvimento de programas

- Adicionar linhas de código

Exemplo:

```
def distancia(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 – y1
    dquadrado = dx**2 + dy**2
    resultado = math.sqrt(dquadrado)
    return resultado
```

Importar o módulo matemático math

- Testar a função novamente

Desenvolvimento de programas

- Adicionamos linhas de código aos poucos
- Desenvolvimento incremental
  - Comece com um programa que funciona
  - Use variáveis temporárias
  - Adicione/remova código muleta

Composição

- Chamar uma função de dentro de outra

- Exemplo: Escrever uma função que recebe dois pontos, o centro de um círculo e um ponto em seu perímetro, e calcula a área do círculo

- Ponto do centro: xc e yc
- Ponto do perímetro: xp e yp
- Encontrar o raio do círculo
- Encontrar área do círculo com esse raio

raio = distancia

resultado = area

Composição

- Ponto do centro: xc e yc
- Ponto do perímetro: xp e yp
- Encontrar o raio do círculo

Exemplo:

```
raio = distancia(xc, yc, xp, yp):
```

- Encontrar área do círculo com esse raio

Exemplo:

```
  resultado = area(raio)
  return resultado
```

Composição

- Escrevendo a função

Exemplo:

```
def area2(xc, yc, xp, yp):
 raio = distancia(xc, yc, xp, yp)
 resultado = area(raio)
 return resultado
```

Repare as expressões area2 e area

Nomes únicos

Variáveis raio e resultado são úteis para depuração (debugging)

Funções booleanas

- Retorna ou *True* ou *False*
- Usa nomes que soem como perguntas sim/não

Exemplo:

```
def ehDivisivel(x, y):
  If  x % y == 0:
    return True   #é verdadeiro (True), é divisível
  else:
    return False  #é falso (False), não é divisível
```

Funções booleanas

- Tornar a função mais concisa se tirarmos vantagem do fato de a condição da instrução *if* ser ela mesma uma expressão booleana

Exemplo:

```
def ehDivisivel(x, y):
  return x % y == 0
```

Exemplo:

```
>>> ehDivisivel(6, 4)
False
>>> ehDivisivel(6, 3)
True
```

Funções booleanas

- Frequentemente usadas em comandos condicionais:

Exemplo:

```
if ehDivisivel(x, y):
  print ("x é divisível por y")
else:
  print ("x não é divisível por y")
```

Mais recursividade

- Definição recursiva é similar à uma definição circular: faz referência à coisa que está sendo definida

Função matemática fatorial

0! = 1

n! = n.(n-1)!

Exemplo:

```
def fatorial(n):
  if n == 0:
   return 1
```

Resultado imediato

Mais recursividade

Função matemática fatorial

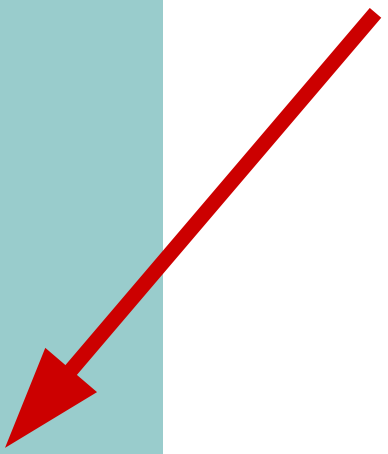
0! = 1

n! = n.(n-1)!

Exemplo:

```
def fatorial(n):
  if n == 0:
    return 1
  else:
    recursivo = fatorial(n-1)
    resultado = n * recursivo
    return resultado
```

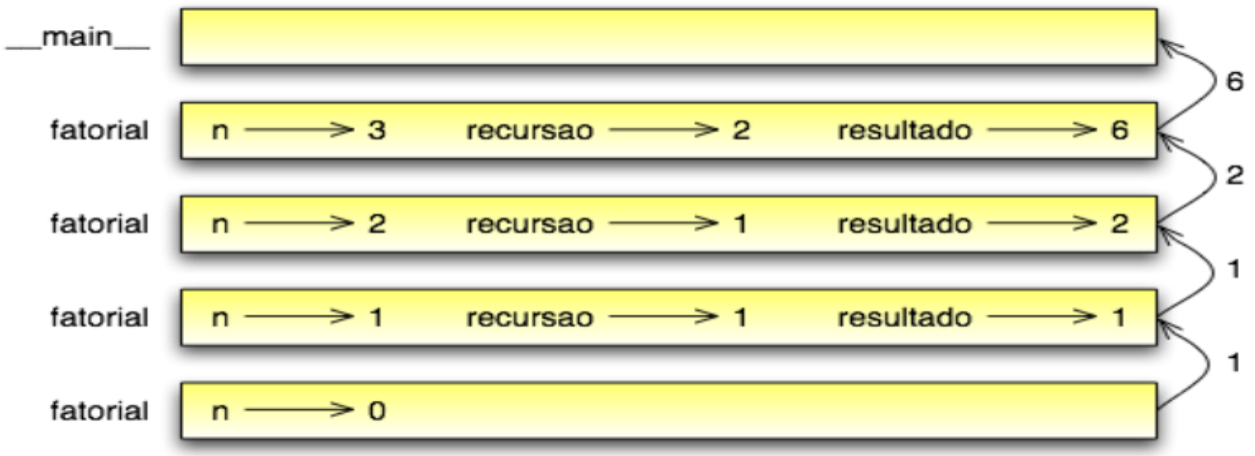
chamada recursiva para encontrar o fatorial de n-1 e multiplicá-lo por n



Mais recursividade

Exemplo:

```
def fatorial(n):
  if n == 0:
    return 1
  else:
    recursivo = fatorial(n-1)
    resultado = n * recursivo
    return resultado
```



Voto de confiança (*Leap of faith*)

- Você assume que a função funciona corretamente, retorna o valor apropriado e não examina a implementação destas funções
- Exemplos: math.cos ou math.exp
- "Assumindo que eu possa encontrar o fatorial de n-1, posso calcular o fatorial de n?"

Mais um exemplo

- Código claro *versus* código enxuto

Função fibonacci

fibonacci(0) = 1
fibonacci(1) = 1
fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)

Exemplo:

```
def fibonacci(n):
  if n == 0 or n == 1:
    return 1
  else:
    return fibonacci(n-1) + fibonacci(n-2)
```

Checagem de tipos

- O que acontece se chamamos fatorial e damos a ela 1.5 como argumento?

Exemplo:

```
def fatorial(n):
  if n == 0:
    return 1
  else:
    recursivo = fatorial(n-1)
    resultado = n * recursivo
    return resultado
```

Exemplo:

```
>>> fatorial(1.5)
 RuntimeError: Maximum recursion depth exceeded
```

Checagem de tipos

- Função fatorial deve:
  - Funcionar com números em ponto flutuante: função gamma
  - Fazer checagem de tipo dos parâmetros: type

Checagem de tipos

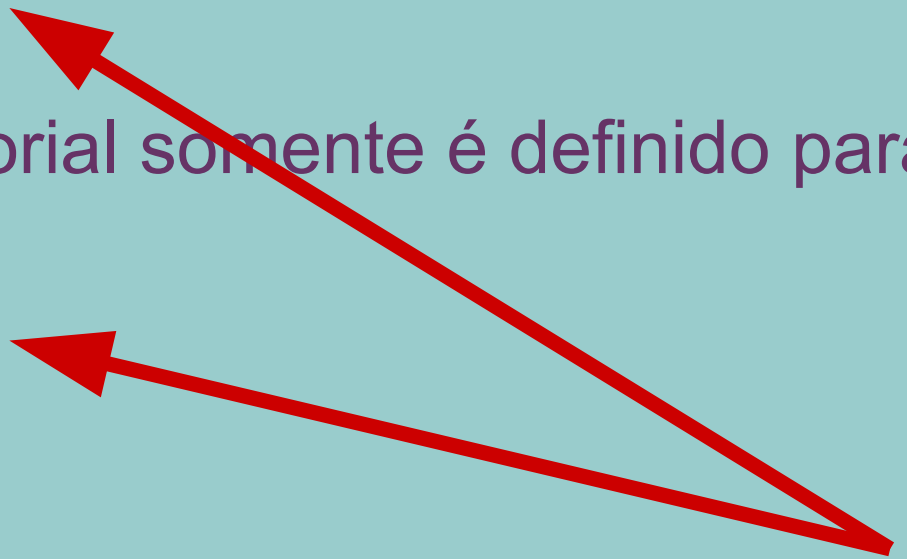
Exemplo:

```
def fatorial (n):
  if type (n) != type (1):
    print ("Fatorial somente é definido para inteiros.")
    return -1
  elif n < 0:
    print ("Fatorial somente é definido para inteiros
positivos")
    return -1
  elif n == 0:
    return 1
  else:
    return n * fatorial(n-1)
```

Checagem de tipos

Exemplo:

```
def fatorial (n):
  if type (n) != type (1):
    print ("Fatorial somente é definido para inteiros.")
    return -1
  elif n < 0:
    print ("Fatorial somente é definido para inteiros
positivos")
    return -1
  elif n == 0:
    return 1
  else:
    return n * fatorial(n-1)
```



Padrão (pattern) guardião

Checagem de tipos

Exemplo:

```
>>> fatorial (“Fred”)
Fatorial somente é definido para inteiros.
-1
>>> fatorial (- 2)
 Fatorial somente é definido para inteiros positivos.
-1
```

Referências

- Aprenda Computação com Python 3.0, Versão 1. Allen Downey, Jeff Elkner and Chris Meyers
  - Capítulo 5: Funções frutíferas