Introdução à Ciência da Computação

Disciplina: 113913

Prof. Edison Ishikawa

Python 3.0

Capítulo 3

Funções

UnB | CIC

Sumário

- Chamadas de funções
- Conversão entre tipos
- Coerção entre tipos
- Funções matemáticas
- Composição
- Adicionando novas funções
- Definições e uso
- Fluxo de execução
- Parâmetros e argumentos
- Variáveis e parâmetros são locais
- Diagramas da pilha
- Funções com resultados

Chamadas de funções

Exemplo de uma chamada de função:

```
>>> type('32')
<class 'str'>
```

- O nome da função é type e ela exibe o tipo de um valor ou variável.
- O valor ou variável, que é o argumento da função, tem que vir entre parênteses.
- Uma função 'recebe' um valor ou mais valores e 'retorna' um resultado.
- O resultado é chamado de valor de retorno.

Chamadas de funções

Exemplo de uma função que atribui valor de retorno a uma variável:

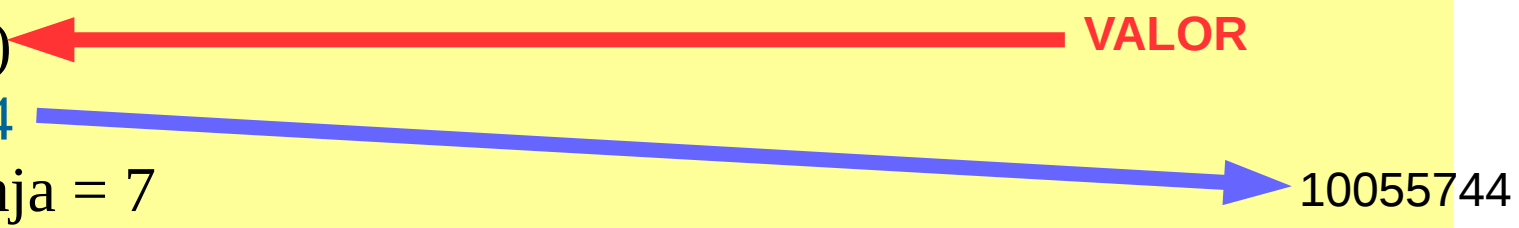
```
>>> retorno_tipo = type('32')
>>> print (retorno_tipo)
<class 'str'>
```

Chamadas de funções

Memória

Outro exemplo de uma função que atribui valor de retorno a uma variável

```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>> id(7)
10055744
>>> laranja = 7
>>> id (laranja)
10055744
```



- Todo valor tem um id, que é um número único relacionado ao local onde ele está guardado na memória do computador. O id de uma variável é o id do valor a qual ela se refere.

Conversão entre tipos

- Python provê uma coleção de funções nativas que convertem valores de um tipo em outro.
- A função int recebe um valor e o converte para inteiro, se possível, ou, se não, reclama

Exemplo

```
>>> int('32')
32
>>> int('Alô')
ValueError: invalid literal for int() : Alô
```

Conversão entre tipos

- int também pode converter valores em ponto flutuante para inteiro, mas lembre que isso trunca a parte fracionária

Exemplo

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

Conversão entre tipos

- A função float converte inteiros e strings em números em ponto flutuante:

Exemplo

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

Conversão entre tipos

- A função str converte para o tipo string

Exemplo

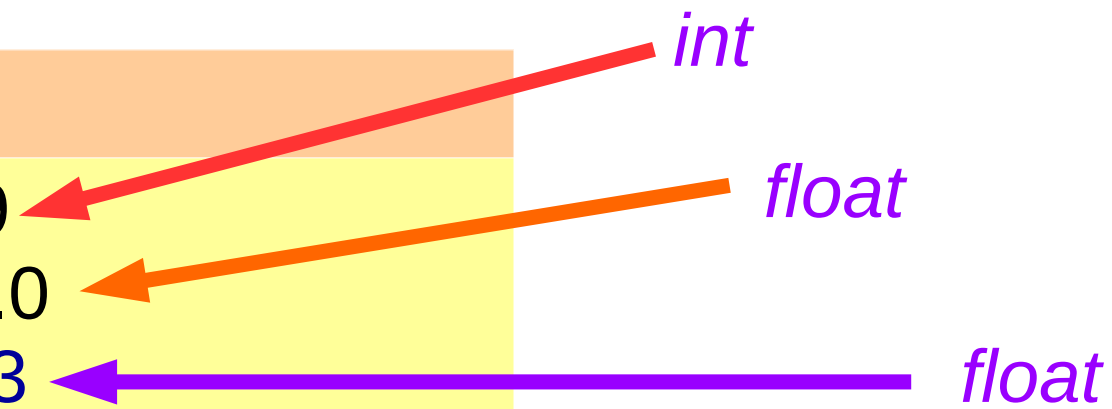
>>> str(32)
’32’
>>> str(3.14149)
’3.14149’

Coerção entre tipos

- Para os operadores matemáticos, se qualquer operando for um *float*, o outro é automaticamente convertido para *float*

Exemplo

```
>>> minuto = 59
>>> minuto / 60.0
0.983333333333
```



- *Fazendo o denominador um float, forçamos o Python a fazer a divisão em ponto flutuante.*

Funções Matemáticas

- Em matemática, você provavelmente já viu funções como seno (sen, sin em inglês) e logaritmo (log), e aprendeu a resolver expressões como sen(pi/2) e log(1/x).
- Primeiro você resolve e expressão entre parênteses (o argumento). Por exemplo, pi/2 é aproximadamente 1,571, e 1/x é 0.1 (se x for 10,0).
- Aí você avalia a função propriamente dita, seja procurando numa tabela ou realizando vários cálculos.
  - O sen de 1,571 é 1 e o log de 0,1 é -1 (assumindo que log indica o logaritmo na base 10).
- Este processo pode ser aplicado repetidamente para avaliar expressões mais complicadas, como log(1/sen(pi/2)).
  - Primeiro você avalia o argumento na função mais interna, depois avalia a função e assim por diante.

Funções Matemáticas

- Python tem um módulo matemático que provê a maioria das funções matemáticas mais familiares.
- Um módulo é um arquivo que contém uma coleção de funções relacionadas agrupadas juntas.
- Antes de podermos usar as funções contidas em um módulo, temos de importá-lo

Exemplo
>>> import math

Funções Matemáticas

- Para chamar uma das funções, temos que especificar o nome do módulo e o nome da função, separados por um ponto.
- Esse formato é chamado de notação de ponto:

Exemplo

```
>>> import math
>>>
>>> decibel = math.log10(17.0)
>>> angulo = 1.5
>>> altura = math.sin(angulo)
```

Funções Matemáticas

- Cuidado, nas funções trigonométricas o ângulo é em radianos

Exemplo

```
>>> graus = 45
>>> angulo = graus * 2 * math.pi / 360.0
>>> math.sin(angulo)
0.707106781187
```

Composição

- Do mesmo modo como nas funções matemáticas, as funções do Python podem ser compostas, o que significa que você pode usar uma expressão como parte de outra.

Exemplo

```
>>> x = math.cos(angulo + math.pi/2)
>>>
>>> x = math.exp(math.log(10.0))
```

Adicionando novas funções

- Criar novas funções para resolver seus próprios problemas é uma das coisas mais úteis de uma linguagem de programação de propósito geral.
- No contexto de programação, função é uma sequência nomeada de instruções ou comandos, que realizam uma operação desejada.
- Esta operação é especificada numa definição de função.
- Até agora, as funções que vimos são pré-definidas e suas definições não foram apresentadas.
  - Isso demonstra que podemos usar funções sem ter que nospreocupar com os detalhes de suas definições.

Adicionando novas funções

- A sintaxe para uma definição de função é:

```
def NOME_DA_FUNCAO( LISTA DE PARAMETROS ) :
    COMANDOS
```

- Você pode usar o nome que quiser para as funções que criar, exceto as palavras reservadas do Python.
- A lista de parâmetros especifica que informação, se houver alguma, você tem que fornecer para poder usar a nova função.
- Uma função pode ter quantos comandos forem necessários

Adicionando novas funções

- As primeiras funções que vamos mostrar não terão parâmetros, então, a sintaxe terá esta aparência:

Exemplo

```
def novaLinha():
        print ()
```

indentação

Adicionando novas funções

Exemplo de script usando a função criada

```
print (’Primeira Linha.’)
novaLinha( )
print (’Segunda Linha.’)
```

Saída do script

```
Primeira Linha.

Primeira Linha.
```

Adicionando novas funções

Exemplo de script usando a função criada

```
print (’Primeira Linha.’)
novaLinha( )
novaLinha( )
novaLinha( )
print (’Segunda Linha.’)
```

Saída do script

```
Primeira Linha.



Primeira Linha.
```

Adicionando novas funções

- Ou poderíamos escrever uma nova função chamada tresLinhas, que produzisse três novas linhas:

Exemplo

```
def tresLinhas( ) :
        novaLinha( )
        novaLinha( )
        novaLinha( )

print (’Primeira Linha.’)
TresLinhas( )
print (’Segunda Linha.’)
```

- Esta função contém três comandos, todos com recuo de quatro espaços a partir da margem esquerda.
- Já que o próximo comando não está endentado, Python reconhece que ele não faz parte da função.

Adicionando novas funções

- Algumas coisas que devem ser observadas sobre este programa:

  1) Você pode chamar o mesmo procedimento repetidamente.
     - Isso é muito comum, além de útil.
  2) Você pode ter uma função chamando outra função; neste caso tresLinhas chama novaLinha.

- Pode não estar claro, até agora, de que vale o esforço de criar novas funções - existem várias razões, mas este exemplo demonstra duas delas:
  - Criar uma nova função permite que você coloque nome em um grupo de comandos. As funções podem simplificar um programa ao ocultar a execução de uma tarefa complexa por trás de um simples comando cujo nome pode ser uma palavra em português, em vez de algum código misterioso.
  - Criar uma nova função pode tornar o programa menor, por eliminar código repetido. Por exemplo, um atalho para 'imprimir' nove novas linhas consecutivas é chamar tresLinhas três vezes.

Definições e Uso

- Reunindo os fragmentos de código o programa completo fica assim:

Exemplo

```
def novaLinha() :
    print ()

def tresLinhas() :
    novaLinha()
    novaLinha()
    novaLinha()

print (’Primeira Linha.’)
tresLinhas()
print (’Segunda Linha.’)
```

Definições e Uso

- Esse programa contém duas definições de funções: novaLinha e tresLinhas.
- Definições de funções são executadas como quaisquer outros comandos, mas o efeito é criar a nova função.
- Os comandos dentro da definição da função não são executados até que a função seja chamada, logo, a definição da função não gera nenhuma saída.
- Como você já deve ter imaginado, é preciso criar uma função antes de poder executá-la.
- Em outras palavras, a definição da função tem que ser executada antes que ela seja chamada pela primeira vez.

Parâmetros e argumentos

- Dentro da função, os valores que lhe são passados são atribuídos a variáveis chamadas parâmetros.

Exemplo de uma função definida pelo usuário, que recebe um parâmetro

```
def imprimeDobrado(parametro1):
        print (parametro1, parametro1)
```

Parâmetros e argumentos

- Esta função recebe um único argumento e o atribui a um parâmetro chamado parametro1

```
def imprimeDobrado(parametro1):
        print (parametro1, parametro1)
```

```
>>> imprimeDoobrado('Spam')
Spam Spam
>>> imprimeDobrado(5)
5 5
>>> imprimeDobrado(3.14159)
3.14159 3.14159
```

- Na primeira chamada da função, o argumento é uma string. Na segunda, é um inteiro. Na terceira é um float.

Parâmetros e argumentos

- As mesmas regras de composição que se aplicam a funções nativas também se aplicam às funções definidas pelo usuário, assim, podemos usar qualquer tipo de expressão como um argumento para imprimeDobrado:

Exemplo

```
>>> imprimeDobrado('Spam'*4)
SpamSpamSpamSpam SpamSpamSpamSpam
>>> imprimeDobrado(math.cos(math.pi))
-1.0 -1.0
```

Parâmetros e argumentos

- Também podemos usar uma variável como argumento

Exemplo

```
>>> aluno_UnB = 'Eric, the half a bee.'
>>> imprimeDobrado(aluno_UnB)
Eric, the half a bee. Eric, the half a bee.
```

- N.T.: “Eric, the half a bee” é uma música do grupo humorístico britânico Monty Python. A linguagem Python foi batizada em homenagem ao grupo e, por isso, os programadores gostam de citar piadas deles em seus exemplos.

Variáveis e parâmetros são locais

- Quando você cria uma variável local dentro de uma função, ela só existe dentro da função e você não pode usá-la fora de lá.

Exemplo

```
def concatDupla(parte1, parte2)
        concat = parte1 + parte2
        imprimeDobrado(concat)
```

Variáveis e parâmetros são locais

- Esta função recebe dois argumentos, concatena-os, e então imprime o resultado duas vezes. Podemos chamar a função com duas strings:

Exemplo

```
>>>canto1 = ’Pie Jesu domine, ’
>>>canto2 = ’dona eis requiem. ’
>>>concatDupla(canto1, canto2)
Pie Jesu domine, dona eis requiem. Pie Jesu domine, dona eis requiem.
```

Variáveis e parâmetros são locais

- Quando a função concatDupla termina, a variável concat é destruída. Se tentarmos imprimi-la, teremos um erro:

Exemplo

```
>>> print (concat)
NameError: concat
```

- Parâmetros são sempre locais. Por exemplo, fora da função imprimeDobrado, não existe nada que se chama parametro1. Se você tentar utilizá-la, o Python vai reclamar.

Referências

- Aprenda Computação com Python 3.0, Versão 1. Allen Downey, Jeff Elkner and Chris Meyers