GRADUAÇÃO

UniCesumar

Python

Funções

Aparecido Vilela Junior

Aparecido.vilela@unicesumar.edu.br

Funções Simples

- Funções podem ser criadas em python de uma forma bastante similar às funções matemáticas.
- A forma geral é:

```
def (argumentos):
    código
    return resultados
```

- Exemplo: $f(x) = x^2$

```
def f (x):
    return x * x
print f(2)
```

Funções

- Chamada de Funções

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

- O nome da função é **type** e ela exibe o tipo de um valor ou variável. O valor ou variável, que é chamado de argumento da função, tem que vir entre **parênteses**.

- É comum se dizer que uma função ‘recebe’ um valor ou mais valores e ‘retorna’ um resultado. O resultado é chamado de valor de retorno.

Funções

- Em vez de imprimir um valor de retorno, podemos atribui-lo a uma variável:

```
>>> bia = type('32')
>>> print (bia)
<type 'str'>
```

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:

- >>> 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:

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

Conversão entre tipos

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

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

Conversão entre tipos

- a função **str** converte para o tipo string:
- >>> str(32)
- '32'
- >>> str(3.14149)
- '3.14149'

- Pode parecer curioso que Python faça distinção entre o valor inteiro 1 e o valor em ponto flutuante 1.0.
- Eles podem representar o mesmo número, mas pertencem a tipos diferentes. A razão é que eles são representados de modo diferente dentro do computador.

Coerção entre tipos

- Agora que podemos converter entre tipos, temos outra maneira de lidar com a divisão inteira.
- Suponha que queiramos calcular a fração de hora que já passou. A expressão mais óbvia, minuto / 60, faz aritmética inteira, assim, o resultado é sempre 0, mesmo aos 59 minutos passados da hora.

- Uma solução é converter minuto para ponto flutuante e fazer a divisão em ponto flutuante:

- >>> minuto = 59
- >>> float(minuto) / 60
- 0.983333333333

Coerção entre tipos

- Opcionalmente, podemos tirar vantagem das regras de conversão automática entre tipos, chamada de coerção de tipos. Para os operadores matemáticos, se qualquer operando for um float, o outro é automaticamente convertido para float:

- >>> minuto = 59
- >>> minuto / 60.0
- 0.983333333333

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

Adicionando novas funções

- Até aqui, temos utilizado somente as funções que vêm com Python, mas também é possível adicionar novas funções.

- Criar novas funções para resolver seus próprios problemas é uma das coisas mais úteis de uma linguagem de programação de propósito geral.

Adicionando novas funções

- 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.

- A sintaxe para uma definição de função é:

- def NOME_DA_FUNCAO(LISTA DE PARAMETROS) :
- COMANDOS

Adicionando novas funções

- 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, mas eles precisam ser endentados a partir da margem esquerda.

Adicionando novas funções

- def novaLinha():
- print()

- Esta função é chamada de novaLinha.
- Os parênteses vazios indicam que ela não tem parâmetros. Contém apenas um único comando, que gera como saída um caractere de nova linha (isso é o que acontece quando você usa um comando print sem qualquer argumento).

Adicionando novas funções

- A sintaxe para a chamada desta nova função é a mesma sintaxe para as funções nativas:
  - print ('Primeira Linha.‘)
  - novaLinha()
  - print ('Segunda Linha.‘)
- A saída deste programa é:
  - Primeira Linha.
  - Segunda Linha.

Fluxo de Execução

- A execução sempre começa com o primeiro comando do programa. Os comandos são executados um de cada vez, pela ordem, de cima para baixo.

- As definições de função não alteram o fluxo de execução do programa, mas lembre-se que comandos dentro da função não são executados até a função ser chamada.
- Chamadas de função são como um desvio no fluxo de execução.

- Qual a moral dessa história sórdida? Quando você for ler um programa, não o leia de cima para baixo. Em vez disso, siga o fluxo de execução.

Argumentos

- Veja um exemplo de uma função definida pelo usuário, que recebe um parâmetro:

- def imprimeDobrado(valor):
- print (valor, valor)

- Esta função recebe um único argumento e o atribui a um parâmetro chamado valor.
- O valor do parâmetro (a essa altura, não sabemos qual será) é impresso duas vezes, seguido de uma nova linha. Estamos usando valor para mostrar que o nome do parâmetro é decisão sua, mas claro que é melhor escolher um nome que seja mais ilustrativo.

Argumentos

- imprimeDobrado('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.

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:

- >>> imprimeDobrado('Spam'*4)
- SpamSpamSpamSpam SpamSpamSpamSpam

Variáveis e Parâmetros 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. Por exemplo:
  - def concatDupla(parte1, parte2):
  - concat = parte1 + parte2
  - imprimeDobrado(concat)

- Esta função recebe dois argumentos, concatena-os, e então imprime o resultado duas vezes. Podemos chamar a função com duas strings:
  - >>> canto1 = Aula, '
  - >>> canto2 = 'Progamação II. '
  - >>> concatDupla(canto1, canto2)

- Quando a função concatDupla termina, a variável concat é destruída. Se tentarmos imprimi-la, teremos um erro:
  - >>> print(concat)
  - NameError: concat
- Parâmetros são sempre locais. Por exemplo, fora da função imprimeDobrado, não existe nada que se chama valor. Se você tentar utilizá-la, o Python vai reclamar.

Exercícios

- Revisão

- Faça um programa para imprimir:
- 1
- 1 2
- 1 2 3
- .....
- 1 2 3 ... n

Resolução

```
def exercicio2(n):
    for i in range(1, n+1):
        for j in range(1,i+1):
            print(j,end=' ')
        print()
```