

Variáveis, Expressões e Comandos

Prof. Alberto Costa Neto
Programação em Python

Constantes

Valores fixos tais como números, letras, e strings são chamados “**constantes**” - porque seus valores não mudam.

- **Constantes** numéricas são como você espera, exceto que os números reais seguem o padrão americano (ponto decimal)
- **Constantes** string usam apóstrofo (') ou aspas (")

```
>>> print 123
```

```
123
```

```
>>> print 98.6
```

```
98.6
```

```
>>> print 'Hello world'
```

```
Hello world
```

Variáveis

Uma **variável** é uma posição na memória que recebe um nome e podem armazenar dados. Este nome de **variável** pode ser usado para recuperar os dados.

- Programadores têm que escolher os nomes das **variáveis**
- Você pode mudar o conteúdo de uma **variável** em comandos posteriores à criação da mesma

x = 12.2

y = 14

x

12.2

y

14

Variáveis

Uma **variável** é uma posição na memória que recebe um nome e podem armazenar dados. Este nome de **variável** pode ser usado para recuperar os dados.

- Programadores têm que escolher os nomes das **variáveis**
- Você pode mudar o conteúdo de uma **variável** em comandos posteriores à criação da mesma

x = 12.2

y = 14

x = 100

x

~~12.2~~ **100**

y

14

Regras para nomes de variáveis em Python

- Devem começar com uma letra ou sublinhado _
- Deve consistir de letras, números e sublinhados
- Sensitivo a caixa (diferencia maiúsculas de minúsculas)
- **Válidos:** spam eggs spam23 _speed
- **Inválidos:** 23spam #sign var.12
- **Diferentes:** spam Spam SPAM

Palavras Reservadas

- Você não pode usar **palavras reservadas** como nomes de variáveis / identificadores

**and del for is raise assert elif
from lambda return break else
global not try class except if or
while continue exec import
pass yield def finally in print
as with**

Sentenças ou Linhas

x	=	2	←	Comando de Atribuição		
x	=	x	+	2	←	Atribuição com expressão
print	x	←	Comando print			

Variável

Operator

Constante

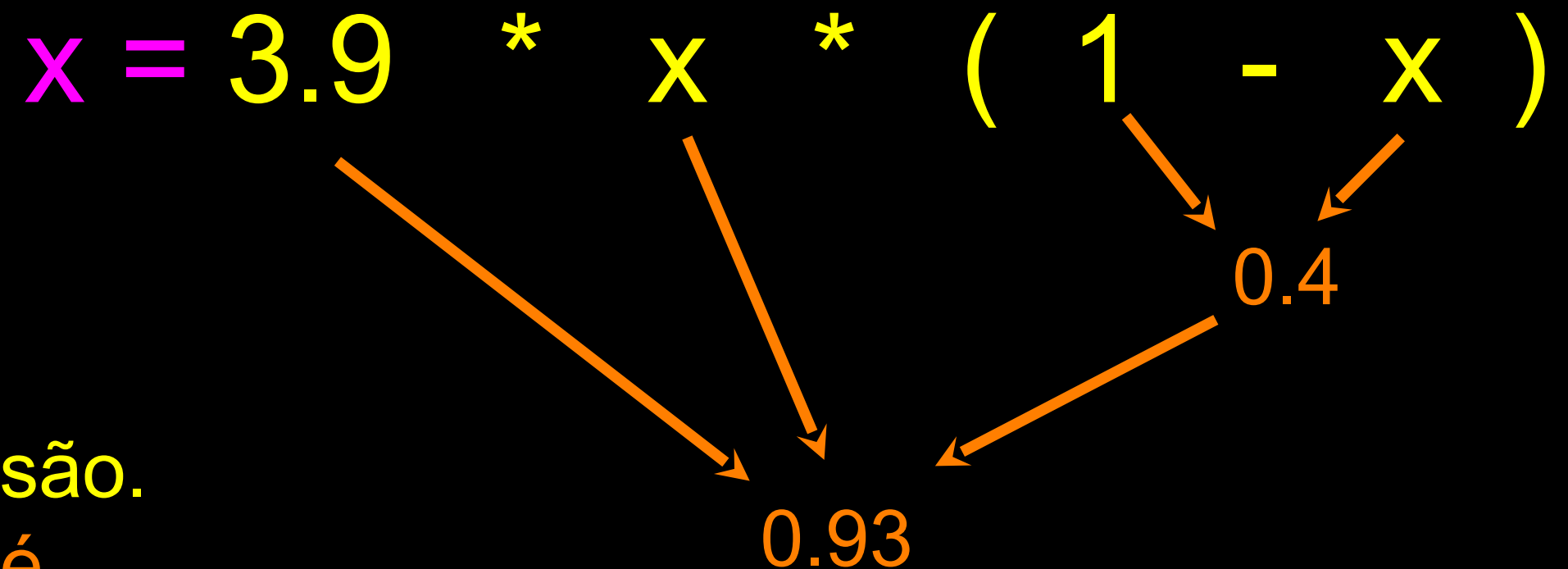
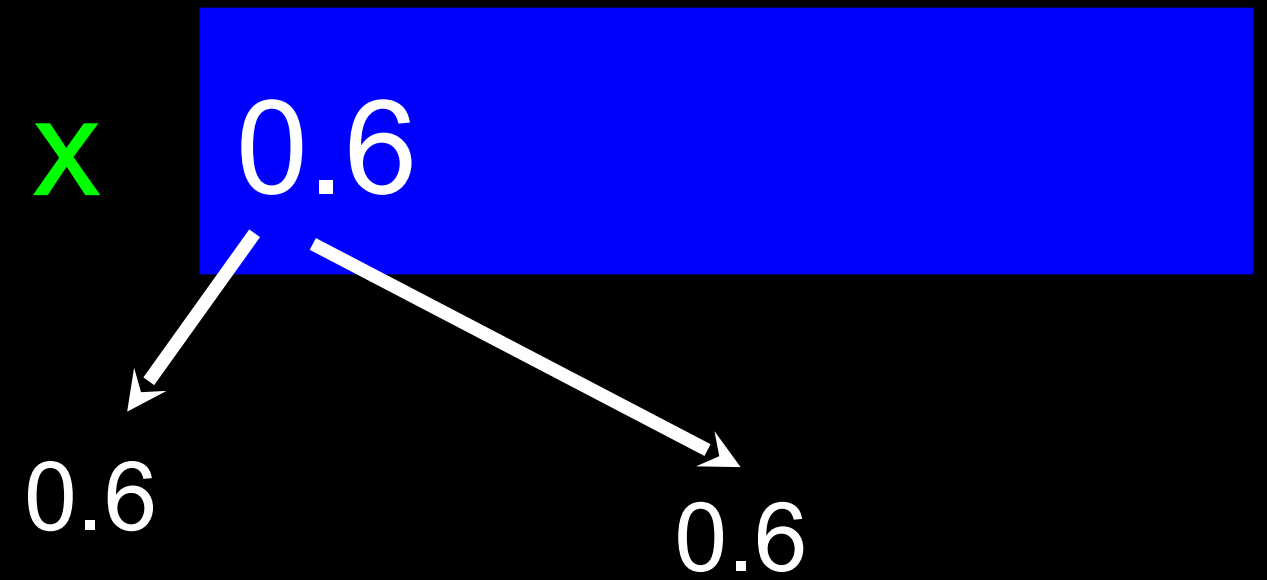
Palavra Reservada

Comando de Atribuição

- Nós atribuímos um valor a uma variável usando o comando de atribuição (=)
- Um comando de atribuição consiste de uma expressão do lado direito e uma variável para armazenar o resultado

$x = 3.9 * x * (1 - x)$

Uma variável é uma posição na memória usada para armazenar um valor (0.6)



O lado direito é uma expressão.
Assim que a expressão é
avaliada, o resultado é
armazenado (atribuído a) x .

Uma variável é uma posição na memória usada para armazenar um valor. O valor armazenado na variável pode ser atualizado, substituindo o valor antigo (0.6) por um novo valor (0.93).

x

~~0.6~~

0.93

$$x = 3.9 * x * (1 - x)$$

O lado direito é uma expressão. Assim que a expressão é avaliada, o resultado é armazenado na (atribuído à) variável no lado esquerdo (isto é, x).

0.93

Expressões Numéricas

- Devido à falta de símbolos matemáticos no teclado do computador, alguns símbolos diferentes foram escolhidos para expressar as operações matemáticas clássicas
- Asterisco é multiplicação
- Exponenciação (elevar a uma expoente) diferencia-se bastante de como escrevemos em matemática

Operador	Operação
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
**	Exponenciação
%	Resto da Divisão

Expressões Numéricas

```
>>> xx = 2
>>> xx = xx + 2
>>> print xx
4
>>> yy = 440 * 12
>>> print yy
5280
>>> zz = yy / 1000
>>> print zz
5
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print kk
3
>>> print 4 ** 3
64
```

$$\begin{array}{r} 4 \text{ R } 3 \\ 5 \overline{) 23} \\ \underline{20} \\ 3 \end{array}$$

Operador	Operação
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
**	Exponenciação
%	Resto da Divisão

Ordem de Avaliação

- Quando usamos vários operadores juntos em uma expressão, Python deve saber qual avaliar primeiro
- Isto é chamado de “precedência de operadores”
- Qual operador “tem precedência” sobre os outros?

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

Regras de Precedência de Operadores

Regra de precedência da maior para a menor:

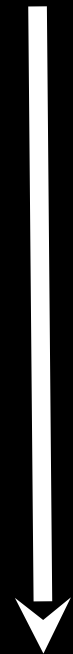
- > Parênteses são sempre respeitados
- > Exponenciação (elevar a um expoente)
- > Multiplicação, Divisão e Resto da Divisão
- > Soma e Subtração
- > Da esquerda para Direita

Parênteses
Exponenciação
Multiplicação
Soma
Esquerda → Direita



```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print x
11
>>>
```

Parênteses
Exponenciação
Multiplicação
Soma
Esquerda → Direita



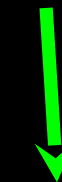
1 + 2 ** 3 / 4 * 5



1 + 8 / 4 * 5



1 + 2 * 5



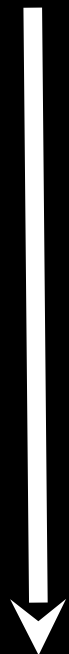
1 + 10



11

Precedência de Operadores

Parênteses
Exponenciação
Multiplicação
Soma
Esquerda → Direita



- Lembre-se das regras de cima para baixo
- Quando escrever código fonte, use parênteses
- Quando escrever código fonte, mantenha as expressões matemáticas simples o bastante para serem fáceis de entender
- Quebre séries longas de operações automáticas para torná-las mais claras

Questão: $x = 1 + 2 * 3 - 4 / 5$

Divisão de Inteiros em Python é estranha!

- Divisão de inteiros trunca os valores
- Divisão entre valores Ponto Flutuante (reais) produzem números em Ponto Flutuante (reais)

```
>>> print 10 / 2  
5
```

```
>>> print 9 / 2  
4
```

```
>>> print 99 / 100  
0
```

```
>>> print 10.0 / 2.0  
5.0
```

```
>>> print 99.0 / 100.0  
0.99
```

Isto mudou à partir de Python 3.0

Misturando Inteiros e Reais (Ponto Flutuante)

- Quando você executa uma operação aonde um operando é um inteiro e o outro operando é um real, o resultado é um real
- O inteiro é convertido em real antes da operação

```
>>> print 99 / 100
0
>>> print 99 / 100.0
0.99
>>> print 99.0 / 100
0.99
>>> print 1 + 2 * 3 / 4.0 - 5
-2.5
>>>
```

Qual é o significado de “Tipo” ?

- Variáveis, Literais e Constantes em Python têm um “tipo”
- Python sabe a diferença entre um número inteiro e uma string
- Por exemplo, “+” significa “somar” se os operandos são números e “concatenar” se for string

```
>>> ddd = 1 + 4
>>> print ddd
5
>>> eee = 'hello ' + 'there'
>>> print eee
hello there
```

concatenar = juntar, unir

Questões de Tipo

- Python sabe o tipo “**type**” de tudo
- Algumas operações são proibidas
- Você não pode “adicionar 1” a uma string
- Podemos perguntar a Python qual é o tipo de qualquer coisa através da função **type()**

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> type(eee)
<type 'str'>
>>> type('hello')
<type 'str'>
>>> type(1)
<type 'int'>
>>>
```

Vários Tipos de Números

- Números têm dois tipos principais
 - > Inteiros (**int**) are números inteiros:
-14, -2, 0, 1, 100, 401233
 - > Números em Ponto Flutuante (**float**)
são números decimais:
-2.5 , 0.0, 98.6, 14.0
- Existem outros tipos de números –
São variações sobre **int** e **float**

```
>>> xx = 1
>>> type (xx)
<type 'int'>
>>> temp = 98.6
>>> type(temp)
<type 'float'>
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
>>>
```

Conversões de Tipos

- Quando você coloca um inteiro e um real em uma expressão, o inteiro é **implicitamente** convertido em um real (**float**)
- Você pode controlar isto com as **funções built-in** (funções que já vêm no Python) **int()** and **float()**

```
>>> print float(99) / 100
0.99
>>> i = 42
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print f
42.0
>>> type(f)
<type 'float'>
>>> print 1 + 2 * float(3) / 4 - 5
-2.5
>>>
```

Conversões de String

- Você pode usar também `int()` e `float()` para converter entre strings e inteiros/reais
- Você receberá um erro (**error**) se a string não contiver caracteres numéricos

```
>>> sval = '123'
>>> type(sval)
<type 'str'>
>>> print sval + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str'
and 'int'
>>> ival = int(sval)
>>> type(ival)
<type 'int'>
>>> print ival + 1
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

Entrada do Usuário

- Você pode instruir o Python a pausar e ler dados providos pelo usuário usando a função `raw_input()`
- A função `raw_input()` retorna uma string

```
nome = raw_input('Quem é você?')  
print 'Bem vindo', nome
```

Quem é você?

Alberto

Bem vindo Alberto

Convertendo a Entrada do Usuário



- Se nós queremos ler um número de um usuário, nós devemos convertê-lo de uma string para um número usando uma **função de conversão de tipo**
- Posteriormente, iremos lidar com dados de entrada inválidos

```
inp = raw_input('Térreo Europeu?')  
teua = int(inp) + 1  
print 'Térreo EUA', teua
```

Térreo Europeu? 0
Térreo EUA 1

Comentários em Python

- Qualquer caractere depois de # é ignorado pelo Python
- Por que comentar?
 - > Descrever o que irá acontecer em uma sequência de código
 - > Documentar quem escreveu o código ou outras informações auxiliares
 - > Desabilitar uma linha de código – Talvez temporariamente

```
# Lê o nome do arquivo e o abre
nome = raw_input('Nome do arquivo:')
arquivo = open(nome, 'r')
texto = arquivo.read()
palavras = texto.split()

# Computa a frequência das palavras no arquivo
contadores = dict()
for palavra in palavras:
    contadores[palavra] = contadores.get(palavra,0) + 1
maior_contador = None
palavra_mais_frequente = None

# Encontra a palavra que aparece com mais frequência
for palavra,contador in contadores.items():
    if maior_contador is None or contador > maior_contador:
        palavra_mais_frequente = palavra
        maior_contador = contador

# Exibe o resultado do processamento do arquivo
print palavra_mais_frequente, maior_contador
```

Operações com String

- Alguns operadores aplicam-se a strings
 - > + significa “concatenação”
 - > * significa “concatenação múltipla”
- Python sabe quando está lidando com uma string ou um número e comporta-se de forma apropriada

```
>>> print 'abc' +  
'123'  
abc123  
>>> print 'Hi' * 5  
HiHiHiHiHi  
>>>
```

Nomes de Variáveis

Mnemônicos

- Como nós programadores temos o poder de escolher os nomes das variáveis, existem algumas “boas práticas
- Escolhemos nomes de variáveis para nos ajudar a lembrar o quê pretendemos armazenar nelas (“**mnemônico**” = “fácil de ser memorizado”)
- Isto pode ser confuso para os iniciantes porque variáveis com nomes bem escolhidos podem parecer tão bons que devem ser palavras chaves

<http://pt.wikipedia.org/wiki/Mnemónica>

```
x1q3z9ocd = 35.0  
x1q3z9afd = 12.50  
x1q3p9afd = x1q3z9ocd * x1q3z9afd  
print x1q3p9afd
```

```
a = 35.0  
b = 12.50  
c = a * b  
print c
```

O quê estes
trechos de código
fonte estão
fazendo?

```
horas = 35.0  
taxa = 12.50  
pagamento = horas * taxa  
print pagamento
```



Acknowledgements / Contributions

Agradecimentos / Contribuições



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information



These slides were translated and adapted by Alberto Costa Neto (albertocn.sytes.net) of the Federal University of Sergipe