

Python - Variáveis e expressões

Prof. André Backes

Linguagens de programação

- Linguagem de Máquina
 - Computador entende apenas pulsos elétricos
 - Presença ou não de pulso
 - 1 ou 0
- Tudo no computador deve ser descrito em termos de 1's ou 0's (binário)
 - Difícil para humanos ler ou escrever
 - $00011110 = 30$

Linguagens de programação

- Linguagens de Alto Nível
 - Programas são escritos utilizando uma linguagem parecida com a linguagem humana
 - Independente da arquitetura do computador
 - Mais fácil programar
 - Uso de compiladores

Linguagens de programação

- Linguagem *Assembly*
 - Uso de mnemônicos
 - Conjunto de o's e i's é agora representado por um código
 - 10011011 -> ADD

Linguagens de programação

- Linguagem *Assembly* - Problemas
 - Requer programação especial (*Assembly*)
 - Conjunto de instruções varia com o computador (processador)
 - Ainda é muito difícil programar

Linguagens de programação

- Linguagens de Alto Nível
 - Programas são escritos utilizando uma linguagem parecida com a linguagem humana
 - Independente da arquitetura do computador
 - Mais fácil programar
 - Uso de compiladores

Linguagens de programação

- Primórdios
 - Uso da computação para cálculos de fórmulas
 - Fórmulas eram traduzidas para linguagem de máquinas
 - Por que não escrever programas parecidos com as fórmulas que se deseja computar?

Linguagens de programação

- FORTRAN (FORmula TRANsform)
 - Em 1950, um grupo de programadores da IBM liderados por John Backus produz a versão inicial da linguagem;
 - Primeira linguagem de alto nível;
- Várias outras linguagens de alto nível foram criadas
 - Algol-60, Cobol, Pascal, etc

Linguagens de programação

- Porquê tantas linguagens?
 - Diferentes níveis de abstração
 - Nível mais alto
 - mais próximo da formulação dos problemas;
 - facilita a programação, detecção e correção de erros
 - Nível mais baixo
 - mais próximo da máquina;
 - potencialmente mais eficiente

Linguagens de programação

- Porquê tantas linguagens?
 - Diferentes tipos de problemas
 - cálculo numérico: Fortran, C, C++
 - sistemas operacionais: C, C++
 - sistemas críticos: Ada, C, C++
 - sistemas web: Java, JavaScript, Ruby, Python

Linguagens de programação

- Porquê tantas linguagens?
 - Diferentes paradigmas
 - imperativo: Algol, Pascal, C
 - funcional: Lisp, Scheme, ML, OCaml, Haskell
 - lógico: Prolog
 - orientado a objetos: Smalltalk, C++, Java, C#
 - Preferências subjetivas (estilo, elegância, legibilidade)

Linguagem Python

- Linguagem de alto nível
 - Sintaxe simples: fácil de aprender
 - Implementação distribuída como código livre
 - Suporta programação procedimental e orientada a objetos
 - Muitas bibliotecas disponíveis
- Pode ser usada em diferentes sistemas operacionais
 - Windows, Linux, Mac OS, etc.
- Usada no “mundo real”
 - Google, Microsoft, Yahoo!, NASA,
- Site oficial: <http://www.python.org>

Linguagem Python

- Trabalha com um interpretador híbrido
 - O programa Python é traduzido para um código intermédio chamado *byte-code*
 - O *byte-code* é executado por um interpretador especial
- Vantagens
 - fácil de usar interativamente
 - fácil testar e modificar componentes
 - mais eficiente do que um interpretador clássico
- Desvantagem
 - não é tão eficiente como uma linguagem compilada tradicional (C, C++, Fortran, etc)

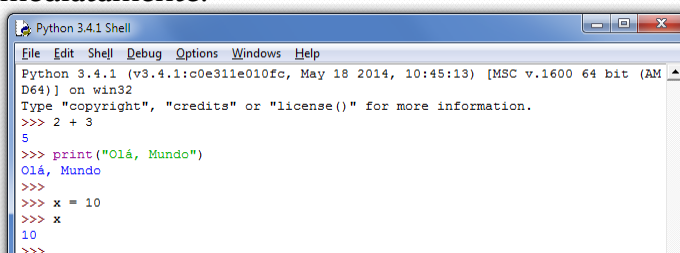
Python - Utilização interativa

- Os comandos são executados no terminal do Python
 - **IDLE** é um ambiente de desenvolvimento integrado para Python

Programas (1)

IDLE (Python GUI)

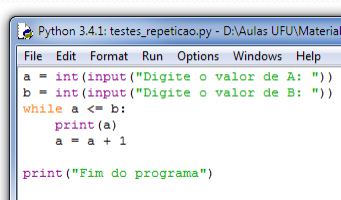
- Podemos escrever comandos Python e ver os resultados imediatamente.



```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 3
5
>>> print("Olá, Mundo")
Olá, Mundo
>>>
>>> x = 10
>>> x
10
>>>
```

Python – Arquivo de script

- Podemos criar um arquivo de texto onde iremos escrever um programa completo (script) e executar de uma só vez pressionando **F5**.
 - Arquivos de programas Python têm extensão **.py**



```
Python 3.4.1: testes_repeticacao.py - D:\Aulas UFU\Material
File Edit Format Run Options Windows Help
a = int(input("Digite o valor de A: "))
b = int(input("Digite o valor de B: "))
while a <= b:
    print(a)
    a = a + 1
print("Fim do programa")
```

Variáveis

- Matemática
 - É uma entidade capaz de representar um valor ou expressão;
 - Pode representar um número ou um conjunto de números
 - $f(x) = x^2$

Variáveis

- Computação
 - Posição de memória que armazena um dado ou valor
 - Pode ser utilizada e modificada pelo programa
 - Deve ser definida antes de ser usada

Variáveis

- Propriedades
 - Nome
 - Pode ter um ou mais caracteres
 - Nem tudo pode ser usado como nome
 - Tipo
 - Conjunto de valores aceitos
 - Escopo
 - global ou local

Variáveis

- Nome
 - Deve iniciar com letras;
 - Podem conter letras, números ou underscores (_). A partir das versões de Python 3.0 as letras podem ser acentuadas;
 - Letras maiúsculas e minúsculas são consideradas diferentes;
 - Palavras reservadas não podem ser usadas como nomes

Variáveis

- Lista de palavras reservadas

and	def	nonLocal	if	not	return
assert	del	finally	import	or	try
break	elif	for	in	pass	while
class	else	from	is	as	yield
continue	except	global	lambda	raise	with
True	False	None			

Variáveis

- Quais nomes de variáveis estão corretos:
 - Contador
 - contador1
 - comp!
 - .var
 - Teste_123
 - _teste
 - int
 - int1
 - 1contador
 - -x
 - Teste-123 x&

Variáveis

- Corretos:
 - Contador, contador1, Teste_123, _teste, int1, int
- Errados
 - comp!, .var,, 1contador, -x, Teste-123, x&

Variáveis

- Tipo
 - Os valores e variáveis em Python são classificados em diferentes tipos
 - O tipo define os valores que a variável pode assumir e as operações que podem ser realizadas com ela
- Diferente de outras linguagens, não precisamos definir o tipo de uma variável
 - O tipo da variável se altera conforme o dado armazenado
 - Comando **type(x)**
 - permite saber o tipo do valor ou variável em x

Variáveis

- Alguns tipos
 - int
 - Tipo inteiro
 - float
 - Números fracionários (ponto flutuante)
 - Na parte decimal usa-se ponto e não vírgula!
 - str
 - String, isto é, cadeia de caracteres
 - Escrevemos o texto entre aspas **'simples'** ou **"duplas"**

```
>>>
>>> x = 1
>>> type(x)
<class 'int'>
>>> y = 1.5
>>> type(y)
<class 'float'>
>>> nome = "Ricardo"
>>> type(nome)
<class 'str'>
>>> type("Oi")
<class 'str'>
>>>
```

Escopo de variáveis

- Escopo
 - Define onde e quando a variável pode ser usada.
- Escopo global
 - A variável é definida fora de qualquer definição de função
 - Tempo de vida é o tempo de execução do programa
- Escopo local
 - A variável é definida dentro de uma função ou na sua lista de parâmetros

Escopo de variáveis

- Exemplo
 - **x** e **y** são variáveis locais
 - Foram definidas em um bloco indentado
 - **x** e **z** são variáveis globais
 - Foram definidas em um bloco não-indentado
 - **x** existe nos dois escopos
 - A variável definida no escopo local ofusca **completamente** o escopo global

Exemplo

```
def func():
    x = 20
    y = 10
    print("Função = ", x, y, z)
```

```
x = 5;
z = 15
print("Antes = ", x, z)
func()
print("Depois = ", x, z)
```

Saída

```
>>>
Antes =  5 15
Função = 20 10 15
Depois =  5 15
>>>
```

Comando de Saída de Dados

- Função **print()**
 - Função que realiza a impressão dos dados do programa no terminal
 - Forma geral
 - `print(expressão1, expressão2, ..., expressãoN)`

Exemplo

```
x = 1
print(x)
print("Teste")
y = 2.5
print("Y = ", y)
```

Saída

```
>>>
1
Teste
Y = 2.5
>>>
```

Comandos de Entrada de Dados

- Função **input()**
 - Função que realiza a leitura de uma cadeia de caracteres do teclado no terminal
 - Forma geral
 - `variável = input(texto)`
 - A função `input()` escreve `texto` no terminal (opcional)

Exemplo

```
str = input("Digite um texto: ")
print("Texto digitado:", str)
```

Saída

```
>>>
Digite um texto: Python
Texto digitado: Python
>>>
```

Comandos de Entrada de Dados

- A função **input()** sempre retorna uma cadeia de caracteres.
 - Mesmo que o que foi digitado contenha apenas números
- O que fazer se for preciso ler um valor numérico?
 - Solução: podemos forçar a **conversão de tipos**

Comandos de Entrada de Dados

- Conversão explícita entre tipos
 - Permite converter um tipo de dado em outro
 - **int(x)**: converte x para inteiro via truncagem (apenas a parte inteira é considerada)
 - **round(x)**: converte x para inteiro via arredondamento
 - **float(x)**: converte x para ponto-flutuante
 - **str(x)**: converte um valor x para texto

```
>>>
>>> str(5.25)
'5.25'
>>>
>>> int(2.5)
2
>>> int("2")
2
>>> round(3.5)
4
>>> float("3.5")
3.5
>>> str(5.25)
'5.25'
>>>
```

Comandos de Entrada de Dados

- Conversão explícita entre tipos
 - Desse modo, podemos combinar o retorno da função **input()** com a conversão de tipos para fazer a leitura de valores numéricos

Exemplo

```
N = int(input("Digite um valor inteiro: "))
N = N * 2
print("O dobro do valor é",N)
```

Saída

```
>>>
Digite um valor inteiro: 5
O dobro do valor é 10
>>>
```

Comentários e docstrings

- Comentários
 - Permitem adicionar uma descrição sobre o programa. Ajudam a lembrar como funciona o programa
 - São ignorados pelo interpretador
 - Começam com o símbolo # e se estendem até ao fim da linha

```
def fatorial(N):
    # Função que calcula o fatorial de um número inteiro N
    # Exemplo: x = fatorial(5)
    if (n <= 1):
        return 1
    return N * fatorial(N-1)
```


Comentários e docstrings

- Docstrings
 - Texto definido entre três aspas duplas ou simples. Similar a um comentário
 - É ignorado pelo interpretador, mas pode ser acessado através do help
 - Normalmente utilizados no início de uma classe, de uma função ou no início do programa

```
def fatorial(N):
    """
    Função que calcula o fatorial de um número inteiro N
    Exemplo: x = fatorial(5)
    """
    if (n <= 1):
        return 1
    return N * fatorial(N-1)
```

Atribuição

- Operador de Atribuição “=”
 - Associa o valor de uma expressão a uma variável
 - A variável é criada neste processo
- Forma geral
 - *nome_da_variável = valor ou expressão*

```
>>>
>>> x = 1
>>> x
1
>>> y = 2.5
>>> y
2.5
>>> a = b = 3
>>> a
3
>>> b
3
>>>
```

Atribuição

- Operador de Atribuição “=”
 - O operador de atribuição “=” armazena o valor ou resultado de uma expressão contida à sua **direita** na variável especificada à sua **esquerda**
 - A linguagem Python suporta múltiplas atribuições

```
>>>
>>> x = 1
>>> x
1
>>> y = 2.5
>>> y
2.5
>>> a = b = 3
>>> a
3
>>> b
3
>>>
```

Atribuição

- Operador de Atribuição “=”
 - É importante notar que a atribuição é um comando, não uma equação
 - Alterar o valor de uma variável não altera o valor das variáveis já calculadas usando o antigo valor

```
>>>
>>> x = 3
>>> y = 2 * x
>>> y
6
>>> x = 5
>>> x
5
>>> y
6
>>>
```

Operadores aritméticos

- Permitem criar expressões aritméticas utilizando números inteiros e fracionários
- Seguem a precedência da matemática: multiplicações e divisões são realizadas antes de soma e subtração

Operador	Descrição	Exemplo
+	Soma	$2 + 3$
-	Subtração	$3 - 1$
*	Multiplicação	$2 * 4$
/	Quociente da Divisão	$4.5 / 2$
**	Exponenciação	$2 ** 3$

Operadores aritméticos

- Podemos alterar a precedência utilizando parênteses ()

```
>>>
>>> 2 + 3 * 2
8
>>> (2 + 3) * 2
10
>>>
```

- O operador de subtração “-” também pode ser utilizado para inverter o sinal de um valor

```
>>>
>>> x = 10
>>> y = -x
>>> x
10
>>> y
-10
>>>
```

Operadores aritméticos

- Alguns operadores são definidos apenas para valores inteiros

Operador	Descrição	Exemplo
//	Quociente da Divisão Inteira	5 // 2
%	Resto da Divisão Inteira	5 // 2

- Exemplos

```
>>>
>>> x = 10 // 3
>>> x
3
>>>
>>> x = 10 % 3
>>> x
1
>>>
```

Operadores aritméticos

- Os operadores aritméticos funcionam com ambos os tipos: **int** e **float**
- Devemos apenas estar atentos ao tipo resultante da operação quando combiná-los
- Operação
 - int + int => int
 - float + float => float
 - int + float => float
 - float + int => float

Operadores aritméticos

- Erros de arredondamento
 - **Valores inteiros:** são representados de forma exata no computador
 - **Valores em ponto-flutuante:** são aproximações finitas dos números reais
- Erros de arredondamento podem se acumular após sucessivas operações sobre estes valores

```
>>>
>>> 8/4
2.0
>>>
>>> 8/3
2.6666666666666665
>>>
```

Operadores aritméticos

- Erros de arredondamento - exemplo
 - Na matemática
 - $(100/3 - 33) * 3$
 - $100 - 33 \times 3 = 1$
 - Usando operações de ponto-flutuante
- O erro de arredondamento foi de
 - $1.0000000000000007 - 1 \approx 7 \times 10^{-15}$

```
>>>
>>> (100/3 - 33) * 3
1.0000000000000007
>>>
>>> 100 - 33 * 3
1
```

Operadores Relacionais

- Permitem a comparação entre os valores de diferentes variáveis

Operador	Descrição	Exemplo
==	Igual	x == 5
!=	Diferente	x != 5
>	Maior do que	x > y
>=	Maior ou igual a	x >= 10
<	Menor do que	y < 100
<=	Menor ou igual a	y <= z

Operadores Relacionais

- Esse tipo de operador retorna **True** (verdadeiro) ou **False** (falso)
 - Podemos utilizar operadores aritméticos durante a comparação

```
>>>
>>> x = 10
>>> y = 20
>>> x == 11
False
>>> x != y
True
>>> 2*x > y
False
>>> 2*x >= y
True
>>>
```

Operadores Lógicos

- São operadores que trabalham com valores lógicos e retornam um valor lógico verdadeiro (1) ou falso (0)

Operador	Descrição	Exemplo
and	Operador "E"	<code>x == 5 and x < y</code>
or	Operador "OU"	<code>x != 5 ou x < 0</code>
not	Operador de negação	<code>not (x > y)</code>

- Exemplos

```
>>> x = 10
>>> y = 20
>>> x > y or x > 0
True
>>> not(x > y)
True
>>> x > y and x > 0
False
>>> not (x > y and x > 0)
True
```

Operadores Lógicos

- Tabela verdade

A	B	not A	not B	A and B	A or B
False	False	True	True	False	False
False	True	True	False	False	True
True	False	False	True	False	True
True	True	False	False	True	True

Operadores de Atribuição Simplificada

- A linguagem Python permite simplificar algumas expressões matemáticas

Operador	Descrição	Exemplo
<code>+=</code>	Soma	<code>c += a</code> equivale a <code>c = c + a</code>
<code>-=</code>	Subtração	<code>c -= a</code> equivale a <code>c = c - a</code>
<code>*=</code>	Multiplicação	<code>c *= a</code> equivale a <code>c = c * a</code>
<code>/=</code>	Quociente da Divisão	<code>c /= a</code> equivale a <code>c = c / a</code>
<code>**=</code>	Exponenciação	<code>c **= a</code> equivale a <code>c = c ** a</code>
<code>//=</code>	Quociente da Divisão Inteira	<code>c //= a</code> equivale a <code>c = c // a</code>
<code>%=</code>	Resto da Divisão Inteira	<code>c %= a</code> equivale a <code>c = c % a</code>

Operadores Bit-a-Bit

- Nas operações bit-a-bit o valor (alto nível) é representado por sua forma binária (baixo nível) e as operações são feitas em cada bit dele

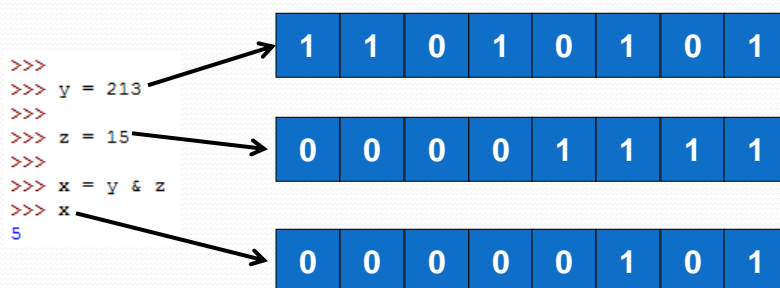
Operador	Descrição	Exemplo
<code>&</code>	E bit-a-bit	<code>x = y & z</code>
<code> </code>	Ou bit-a-bit	<code>x = y z</code>
<code>^</code>	Ou exclusivo	<code>x = y ^ z</code>
<code>~</code>	Complemento bit-a-bit	<code>x = y ~ z</code>
<code><<</code>	Deslocamento a esquerda	<code>x = y << z</code>
<code>>></code>	Deslocamento a direita	<code>x = y >> z</code>

Operadores Bit-a-Bit

- As operações bit-a-bit ajudam programadores que queiram trabalhar com o computador em “baixo nível”
- Essas operações somente podem ser utilizadas em valores inteiros

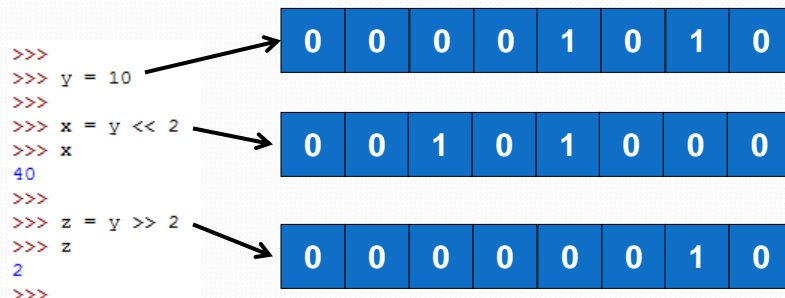
Operadores

- Exemplo de operador bit a bit



Operadores

- Exemplo de operador bit a bit



Precedência dos Operadores

Maior precedência
**
~ + -
* / % //
+ -
>> <<
&
^
<= < > >=
<> == !=
= %= /= //=- += *= **=
is is not
in not in
not or and
Menor precedência

Módulos

- Um módulo Python nada mais é do que um arquivo de extensão `.py` contendo código-fonte Python
 - Este arquivo pode conter variáveis, funções e classes
- A medida que um programa cresce em tamanho e complexidade, um ou mais módulos Python são utilizados de forma combinada

Módulos

- Comando **import**
 - É a instrução mais básica para trabalhar com módulos
 - O módulo deve estar no caminho de procura de módulos do interpretador
 - Alguns dos módulos mais comuns são: **math**, **sys**, **os**, **time**, **random**, **re**, **shelve**
- Forma geral
 - **import nome-módulo**

```
>>>  
>>> import math  
>>>
```

Módulos

- Funções matemáticas
 - Muitas funções e constantes matemáticas estão disponíveis no módulo **math**
- Uso das funções do módulo
 - **nome-módulo.nome-função**

```
>>>
>>> import math
>>> math.e
2.718281828459045
>>>
>>> math.pi
3.141592653589793
>>>
>>> math.sqrt(2)
1.4142135623730951
>>>
```

Módulos

- Função **dir()**
 - **dir()**
 - Retorna uma lista de nomes de todos os símbolos da tabela do módulo atual.
 - **dir(nome-módulo)**
 - Retorna uma lista dos nomes dos atributos contidos em um módulo
 - Permite descobrir quais símbolos e funções o compõem

```
>>> dir()
['_builtins_', '__doc__', '__loader__', '__name__', '__package__', '__spec__',
'math', 'x', 'y', 'z']
>>>
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Material Complementar

- Vídeo Aulas
 - Aula 01 - Introdução
 - <https://youtu.be/GDCtOyEY1jc>
 - Aula 02 - Variáveis
 - <https://youtu.be/PqD3ujwL6GM>
 - Aula 03 - Comandos de Entrada e Saída
 - <https://youtu.be/7LjZqlZtri4>
 - Aula 04 - Comentários
 - <https://youtu.be/PblrCiFllgI>

Material Complementar

- Vídeo Aulas
 - Aula 05 - Atribuição e Operadores aritméticos
 - <https://youtu.be/nRG6VnXy45g>
 - Aula 06 - Operadores Relacionais e Lógicos
 - <https://youtu.be/8-VGKAnMmxw>
 - Aula 07 - Operadores Bit-a-Bit
 - <https://youtu.be/pG55wmEjiW4>
 - Aula 08 - Módulos
 - <https://youtu.be/BbjbyBONVmk>