MC-102 — Aula 02
Shell Interativa, Programa Básico, Variáveis, Atribuições, Tipos Simples

Instituto de Computação – Unicamp

2016

Roteiro

Shell Interativa

- Abra um prompt de comando e execute "python".
- Se Python estiver instalado em seu computador será inicializado a shell de Python.

```
$ python
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Shell Interativa

- Você pode executar comandos diretamente na shell.

```
$ python
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Ola turma")
Ola turma
>>> 5+5
10
>>>
```

Shell Interativa

- A shell é muito útil durante a criação de um programa pois você pode já testar partes do seu código para saber se está funcionando como o esperado.
- Mas na maioria das vezes criaremos um código completo que deve ser salvo em um arquivo com a extensão **.py**.
- Depois este código poderá ser executado em um terminal da seguinte forma

```
$python nomeArquivo.py
```

Estrutura Básica de um Programa em Python

A estrutura básica é a seguinte:

```
Comando1
.
.
.
ComandoN
```

- O programa deve ter um comando por linha.
- Os comandos serão executados nesta ordem, de cima para baixo, um por vez.

Estrutura Básica de um Programa em Python

Exemplo:

```
print("Ola turma de MC102")
print("Vamos programar em Python")
```

Estrutura Básica de um Programa em Python

Exemplo:

```
print("Ola turma de MC102")  print("Vamos programar em Python")
```

Este programa gera um erro pois temos dois comandos em uma mesma linha.

Estrutura Básica de um Programa em Python

Você pode no entanto usar um ponto e vírgula ao final de cada comando para usar vários comandos em uma mesma linha:

```
print("Ola turma de MC102");  print("Vamos programar em Python");
```

- Este programa executa sem problemas.
- Mas neste curso sempre usaremos o padrão de um comando por linha.

Variáveis

Definição

Variáveis são locais onde armazenamos valores. Toda variável é caracterizada por um nome, que a identifica em um programa.

- Durante a execução do programa, um pedacinho da memória corresponde à variável.

Declarando uma variável

Declara-se da seguinte forma: **Nome_Variável = Valor**

- As variáveis em Python são criados dando-se um nome para as mesmas e então atribuindo-se um valor para esta.
- Você só pode usar uma variável se ela for criada antes (vejo exemplo com erro abaixo).

```
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 10
>>> b = 10
>>> a = a+b
>>> a
20
>>> a = a + c
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'c' is not defined
```

Regras para nomes de variáveis

- **Deve** começar com uma letra (maíuscula ou minúscula) ou subcrito(_). **Nunca** pode começar com um número.
- Pode conter letras maiúsculas, minúsculas, números e subscrito.
- Não pode-se utilizar como parte do nome de uma variável:

```
{ ( + - * / \ ; . , ?
```

- Letras maiúsculas e minúsculas são diferentes:

$$c = 4$$
$$C = 3$$

Comando de Atribuição

Definição

O comando de atribuição, denotado pelo sinal =, serve para atribuir valores para variáveis.

- A sintaxe do uso do comando é:

variável = valor

- Exemplos:

```
a = 3
b = 6.57
print(a)
print(b)
```

Constantes

- Constantes são valores que por algum motivo devem aparecer em um programa.
- No programa anterior vimos o uso das constantes 3 e 6.57 utilizados como valores na criação das variáveis **a** e **b**.

```
a = 3
b = 6.57
print(a)
print(b)
```

Comando de Atribuição

- O comando de atribuição pode conter expressões do lado direito:

variável = expressão

- Atribuir um valor de uma expressão para uma variável significa calcular o valor daquela expressão e copiar aquele valor para a variável.

```
a = 3 + 10
b = (6.57 * 90) + 40
print(a)
print(b)
```

Atribuição

- O sinal de igual no comando de atribuição é chamado de **operador de atribuição**.
- Veremos outros operadores mais adiante.

À esquerda do operador de atribuição deve existir somente o nome de uma **variável**.

$=$

À direita, deve haver uma **expressão** cujo valor será calculado e armazenado na variável

Tipos de Variáveis

- As variáveis em Python possuem um tipo, que corresponde ao tipo do valor que está armazenado na variável naquele instante.
- Python não possui tipagem forte como outras linguagens.
  - Isto significa que você pode atribuir valores de diferentes tipos para uma mesma variável.
  - Portanto o tipo da variável pode mudar ao longo da execução do programa!

```
a = 3
print(a)
a = 90.45
print(a)
a = "Ola voces!"
print(a)
```

Tipos de Variáveis em Python

Python possui os seguintes tipos básicos que veremos nesta aula:

- **int**: Corresponde aos números inteiros. Exe: 10, -24.
- **float**: Corresponde aos números racionais. Exe: 2.4142, 3.14159265.
- **string**: Corresponde a textos. Exe: "Ola turma", "Agora vai!".

Os tipos básicos booleanos, bytes, listas, tuplas, conjuntos e dicionários serão vistos ao longo do curso.

Variáveis inteiras

- Em Python dizemos que uma variável é inteira se esta armazena um dado do tipo **int**.
- Em Python2 há um outro tipo de variável inteira cujo tipo é **long**.
- As variáveis do tipo **int** em Python2 possuem valores máximos e mínimos:
  - De $-2^{63} - 1$ até $2^{63} - 1$ (em uma máquina de 64 bits).
  - Em outras linguagens ocorre um problema conhecido como *overflow* (*underflow*) ao tentarmos armazenar um número maior (ou menor) do que o que pode ser representado na variável.
  - Mas Python2 converte automaticamente variáveis **int** para **long** evitando este problema.
- Variáveis *long* possuem precisão arbitrária.
- **Em Python3 só há o tipo int e este possui precisão arbitrária**.

Variáveis inteiras

- O Comando **type** informa o tipo de uma variável.
- Veja como Python2 converte automaticamente uma variável do tipo **int** para **long**.

```
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 900000000000
>>> type(a)
<type 'int'>
>>> a = 1000000000000000*100000000000
>>> type(a)
<type 'long'>
>>>
```

Neste curso usaremos como padrão Python3.

Variáveis de tipo ponto flutuante

Variáveis do tipo **float** armazenam valores reais. Mas possuem problemas de precisão pois há uma quantidade limitada de memória para armazenar um número real. Exemplos de números em ponto flutuante: 2.1345 ou 9098.123.

- **float:** Em computadores de 64bits tem precisão de aproximadamente 15 casas decimais (depois do ponto).
- Pode-se armazenar valores de $2.2250738585072014^{-308}$ até 1.7976931348623157^{308}
- Ao contrário de números inteiros, possuem problemas tanto de precisão quanto de *overflow* (*underflow*).

Variáveis de tipo ponto flutuante

Note o tipo das variáveis, problemas de precisão e problemas de *overflow*.

```
Python 3.4.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 10.0/3.0
>>> a
3.3333333333333335
>>> type(a)
<type 'float'>
>>> a = 10000000000000000.2
>>> a
1e+16
>>> a = a*a*a*a*a
>>> a
1e+80
>>> a = a*a*a*a*a
>>> a
inf
>>>
```

Variáveis de tipo string

Variáveis do tipo **string** armazenam textos. Uma constante do tipo string deve estar entre aspas simples ou aspas duplas. Exemplos de strings: 'Olá Brasil!' ou "Olá Brasil".

```
Python 3.4.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 'Olá Brasil!'
>>> type(a)
<type 'str'>
>>> a
'Olá Brasil!'
>>>
```

Veremos posteriormente neste curso diversas operações que podem ser realizadas sobre dados do tipo **string**.

Variáveis e Constantes

Constantes são valores previamente determinados e que por algum motivo, devem aparecer dentro de um programa.

- Assim como as variáveis, as constantes também possuem um tipo. Os tipos permitidos são exatamente os mesmos das variáveis.
- Exemplos de constantes:

```
85, 0.10, 'c', "Hello, world!"
```

Variáveis e Constantes

- Uma **constante inteira** é um número na forma decimal, como escrito normalmente
  Ex: `10`, `145`, `1000000`
- Uma **constante ponto flutuante** é um número real, onde a parte fracionária vem depois de um ponto
  Ex: `2.3456`, `32132131.5`, `5.0`
- Uma **constante do tipo string** é um texto entre aspas duplas ou aspas simples
  Ex: `"Hello, world!"`

Expressões Simples

Uma constante é uma expressão e como tal, pode ser atribuída a uma variável (ou em qualquer outro lugar onde uma expressão seja necessária).

```
Python 3.4.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 100.456
>>>
```

Expressões Simples

Uma variável também é uma expressão e pode ser atribuída a outra variável.

Ex:

```
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 100.456
>>> b = 2
>>> b = b + a
>>> b
102.456
>>>
```

Exercício

Qual o valor armazenado na variável **a** no fim do programa?

```
d = 3;
c = 2;
b = 4;
d = c + b;
a = d + 1;
a = a + 1;
print(a)
```

Exercício

Você sabe dizer qual erro existe neste programa? Tente rodar o programa abaixo.

```
d = 3.0
c = 2.5
b = 4
d = b + 90
e = c * d
a = a + 1
print(a)
print(e)
```

Informações Extras: Constantes Inteiras

Números inteiros podem ser escritos em outras bases.

- Um número na forma decimal, como escrito normalmente
  Ex: `10`, `145`, `1000000`
- Um número na forma hexadecimal (base 16), precedido de 0x
  Ex: `0xA` ($0xA_{16} = 10$), `0x100` ($0x100_{16} = 256$)
- Um número na forma octal (base 8), precedido de 0
  Ex: `010` ($0x10_8 = 8$)

Informações Extras: Constantes do tipo de ponto flutuante

- Na linguagem Python, um número só pode ser considerado um número decimal se tiver uma parte "não inteira", mesmo que essa parte não inteira tenha valor zero. Utilizamos o ponto para separarmos a parte inteira da parte "não inteira".
  Ex: `10.0`, `5.2`, `3569.22565845`
- Um número inteiro ou decimal seguido da letra **e** mais um expoente. Um número escrito dessa forma deve ser interpretado como:

$$numero \cdot 10^{expoente}$$

Ex: 2e2 ($2e2 = 2 \cdot 10^2 = 200.0$)