MC-102 — Aula 02
Estrutura Básica de um Programa, Variáveis, Dados, Atribuição e Expressões Aritméticas

Instituto de Computação – Unicamp

2019

Shell Interativa

- Abra um terminal 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

- Um programa em Python é uma sequencia de definições e comandos que serão executados pelo interpretador.
- 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.

Objetos/valores

- Um programa executa comandos para manipular informações/dados.
- Qualquer dado em Python é um "objeto" ou valor , que é de um certo **tipo** específico.
- O **tipo** de um objeto/valor especifica quais operações podem ser realizadas sobre o objeto.
- Por exemplo, o número 5 é do tipo **int** em Python.

Variáveis

Definição

Variáveis são uma forma de se associar um nome dado pelo programador com um valor.

- No exemplo abaixo associamos os nomes **altura**, **largura** e **a** com os valores 10, 3, e 29 respectivamente.

```
altura = 10
largura = 3
a = 29
```

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**:

$$\text{casa} = 4$$
$$\text{Casa} = 3$$

Atribuição

O comando = do Python é o comando de atribuição. Ele associa a variável do lado esquerdo do comando com o valor (ou mais corretamente *expressão*) do lado direito do comando.

```
a = 10
b = 11
c = 10
b = 20
```

Atribuição

- Se uma variável for usada sem estar associada com nenhum valor, um erro ocorre.
- No exemplo abaixo não podemos usar a variável **c**, pois esta não foi definida (associada com algum objeto).

```
[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
```

Note o nome do erro: `NameError:` - isso indica uma variavel (`name 'c'`) que nao tem um valor armazenado nela

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 somente depois associar o valor calculado com a variável.

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

Tipos de dados 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.
- **str** ou **string**: Corresponde a textos. Exe: "Ola turma", "Agora vai!".
- **bool**: Corresponde a apenas 2 valores: `True` e `False`. Sao chamados de booleanos

Os outros tipos básicos como listas, tuplas, conjuntos e dicionários serão vistos ao em outras aulas do curso.

Tipo Inteiro

- O Comando **type** informa o tipo de um objeto associado com uma variável.

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information
>>> a = 98
>>> type(a)
<class 'int'>
>>> b = 'ola turma'
>>> type(b)
<class 'str'>
>>>
```

Tipo Inteiro

- Valores do tipo **int** armazenam valores inteiros Exemplos: 3, 1034, e -512.
- O tipo **int** possui precisão arbitrária (limitado à memória do seu computador).

Neste curso usamos como padrão Python3, por isso inteiros possuem precisão arbitrária ao contrário de Python2.

Tipo Ponto Flutuante

- Objetos do tipo **float** armazenam valores "reais".
- Literais do tipo **float** são escritos com um ponto para separar a parte inteira da parte decimal. Exemplos: 3.1415 e 9.8.
- Possuem problemas de precisão pois há uma quantidade limitada de memória para armazenar um número real no computador.
- Notem no exemplo abaixo o erro de precisão:

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information
>>> 1/10.0
0.1
>>> 0.1+0.2
0.30000000000000004
```

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

- Objetos do tipo **string** armazenam textos.
- Um literal 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 objetos do tipo **string**.
String **podem conter** letras com acentos, letras de outras linguas, etc.
Os nomes de variáveis **também** podem conter acentos, letras de outras linguas, mas **recomendamos fortemente** que vc nao faça isso!

Variaveis podem aceitar qualquer dado

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

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)
```

Escrevendo na tela

- Para imprimir um texto, utilizamos o comando **print**.
- O texto pode ser um literal do tipo **string**.

```
print('Ola Pessoal!')
```

- Saída:

```
Ola Pessoal!
```

- No meio da **string** pode-se incluir caracteres de formatação especiais.
- O símbolo especial `\n` é responsável por pular uma linha na saída.

```
print('Ola Pessoal! \n Ola Pessoal!!')
```

- Saída:

```
Ola Pessoal!
Ola Pessoal!!
```

Escrevendo o conteúdo de uma variável na tela

- Podemos imprimir, além de texto puro, o conteúdo de uma variável utilizando o comando **print**.
- Separamos múltiplos argumentos a serem impressos com uma vírgula.

```
a = 10
print('A variavel contem o valor', a)
```

- Saída:

```
A variavel contem o valor 10
```

Escrevendo o conteúdo de uma variável na tela

```
a = 10
b = 3.14
print('a contam o valor', a, '. ja b contem o valor' , b)
```

- A impressão com múltiplos argumentos inclui um espaço extra entre cada argumento. Saída do exemplo:

```
a contem o valor 10 . ja b contam o valor 3.14
```

Formatos ponto flutuante - extra!!!

- Podemos especificar o número de casas decimais que deve ser impresso em um número ponto flutuante usando o especificador **%.Nf**, onde N especifica o número de casas decimais.

```
pi = 3.1415
r = 7
area = pi*r*r
print("Area do circulo de raio", r ," = ",area)
print("Area do circulo de raio %.2f" % r, "= %.2f" % area)
```

- A saída será:

```
Area do circulo de raio 7 = 153.9335
Area do circulo de raio 7.00 = 153.93
```

Exercicio de casa - importante

- A função **print** sempre pula uma linha ao final da impressão. Descubra na interner como usar o print de forma que ele nao mude de linha no final
- a funcao **print** sempre coloca um branco entre os valores que são impressos. Descubra como mudar isso.
- estude como usar esse jeito para formatar valores reais. Isso é chamado de `old style format` em Python. Note que há VARIAS coisas diferentes sobre como escrever esse formato, inclusive o uso do % dentro e fora do string.

A função **input**

- Realiza a leitura de dados a partir do teclado.
- Aguarda que o usuário digite um valor e atribui o valor digitado à uma variável.
- Todos os dados lidos são do tipo string.

```
print("Digite um numero:")
a = input()
print("O numero digitado: ", a)
print(type(a))
```

A função **input**

- Podemos converter uma string lida do teclado em um número inteiro usando a função **int()**.

```
print("Digite um numero:")
a = int(input())
a = a*10
print("O numero digitado vezes 10 e: ", a)
```

A função **input**

- Podemos fazer o mesmo para números ponto flutuante usando a função **float()**.

```
print("Digite um numero:")
a = float(input())
a = a*10
print("O numero digitado vezes 10 e %.2f: " %a)
```

A função **input**

- Nos dois exemplos anteriores é esperado que o usuário digite um número.
- Se o usuário digitar um texto não numérico o programa encerrará com um erro de execução.

```
int("abc")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'abc'
```

Note o nome do erro: `ValueError:` e a razao do erro `invalid literal for int()` e o valor `'abc'`

Expressões

- Constantes e variáveis são expressões.
- Uma expressão também pode ser um conjunto de operações aritméticas, lógicas ou relacionais utilizadas para fazer "cálculos" sobre os valores das variáveis. Exemplo de expressão:

  a + b

  Calcula a soma de **a** e **b**.

  a / 9

  Calcula divisão de **a** e **b** por 9.

Expressões Aritméticas

- Os operadores aritméticos são: +, -, *, /, //, % , **
- Soma: *expressão* + *expressão*

```
>>> 56+9
65
```

- Subtração: *expressão* - *expressão*

```
>>> 56-9
47
```

- Produto *expressão* * *expressão*

```
>>> 56*9
504
```

Expressões Aritméticas

- *expressão* / *expressão* : Calcula a divisão de duas expressões. O resultado é sempre um número ponto flutuante.

```
>>> 27/9
3.0
```

- *expressão* // *expressão* : Calcula a divisão de duas expressões. Se os operandos forem inteiros a divisão é inteira.

```
>>> 5//2
2
>>> 5//2.0
2.0
```

Expressões

No exemplo abaixo, quais valores serão impressos?

```
print(9/2)
print(9//2)
print(9//2.0)
```

Expressões Aritméticas

- *expressão* ** *expressão* : Calcula o valor da expressão à esquerda elevado ao valor da expressão à direita.

```
>>> 2**4
16
>>> 2.2**4
23.425600000000006
```

- *expressão* % *expressão* : Calcula o resto da divisão (inteira) de duas expressões.

```
>>> 5 % 2
1
>>> 9 % 7
2
>>> 2 % 5
2
```

Expressões

No exemplo abaixo, quais valores serão impressos?

```
print(29 % 3)
print(19 % 5)
print(3 % 15)
```

Expressões

- As expressões aritméticas (e todas as expressões) operam sobre outras expressões.
- É possível compor expressões complexas como por exemplo:
  `a = b *( (2 / c)+(9 + d * 8) );`
- Qual o valor da expressão `5 + 10 % 3`?
- E da expressão `5 * 10 % 3`?

Precedência

- Precedência é a ordem na qual os operadores serão avaliados quando o programa for executado. Em Python, os operadores são avaliados na seguinte ordem:
  - **
  - *, /, //, na ordem em que aparecerem na expressão.
  - %
  - + e -, na ordem em que aparecerem na expressão.
- Exemplo: 8+10*6 é igual a 8+60 que é igual a 68

Alterando a precedência

- (*expressão*) também é uma expressão, que calcula o resultado da expressão dentro dos parênteses, para só então calcular o resultado das outras expressões.
  - $5 + 10 \,\%\, 3$ é igual a 6
  - $(5 + 10) \,\%\, 3$ é igual a 0
- Você pode usar quantos parênteses desejar dentro de uma expressão.
- Use sempre parênteses em expressões para deixar claro em qual ordem a expressão é avaliada!

Conversão de Tipos

- Já vimos o uso das funções **int()**, **float()** e **str()** que servem para converter dados do tipo **str** no outro especificado pela função.
- mas eles server para converter qualquer tipo em **int()**, **float()** e **str()**, desde que isso faça sentido.
- A conversão só ocorre se o dado estiver bem formado. Por exemplo **int("aaa")** resulta em um erro.
- Ao convertermos um número **float** para **int** ocorre um truncamento, ou seja, toda parte fracionária é desconsiderada.

```
>>> a = "ola"
>>> int(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'ola'
>>> int(2.99)
2
>>> int(-2.99)
-2
>>> float("3.1415")
3.1415
>>>
```

Exercício

- Crie um programa que:
  - Lê uma string, pula uma linha e imprime a string lida.
  - Lê um inteiro, pula uma linha e imprime o inteiro lido.
  - Lê um número ponto flutuante, pula uma linha e imprime o número lido.

Exercício

- Crie um programa que lê dois números reais e que computa e imprime a soma, a diferença, a multiplicação e divisão dos dois números.
- Crie um programa que le o numero de centavos (um inteiro) e diz quantos como pagar esse numero de centavos usando notas de 100, 50, 20, 10 e 2 reais, moedas de 1 real, e moedas de 50, 25, 10, 5 e 1 centavos.
- Crie um programa que le o numero de segundos (um inteiro) e diz quantos dias, horas, minutos e segundos sao essa quantidade de segundos

Proxima aula

Booleanos, comparacoes, expressoes logicas e **if-then-else**

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 decimal.
  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` ($2e2 = 2 \cdot 10^2 = 200.0$)