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

Prof. Luiz F. Bittencourt

Turmas QR

Instituto de Computação – Unicamp

2019

Conteúdo adaptado de slides fornecidos pelo Prof. Eduardo Xavier.

Roteiro

Shell Interativa

- Abra um terminal de comando e execute “python”.
- Se Python estiver instalado em seu computador será inicializada a *shell* 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 esperado.
- Entretanto, na maioria das vezes criaremos um código completo que deve ser salvo em um arquivo com a extensão **.py**.
- 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 ter vários comandos em uma mesma linha:

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

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

Objetos

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

Variáveis

Definição

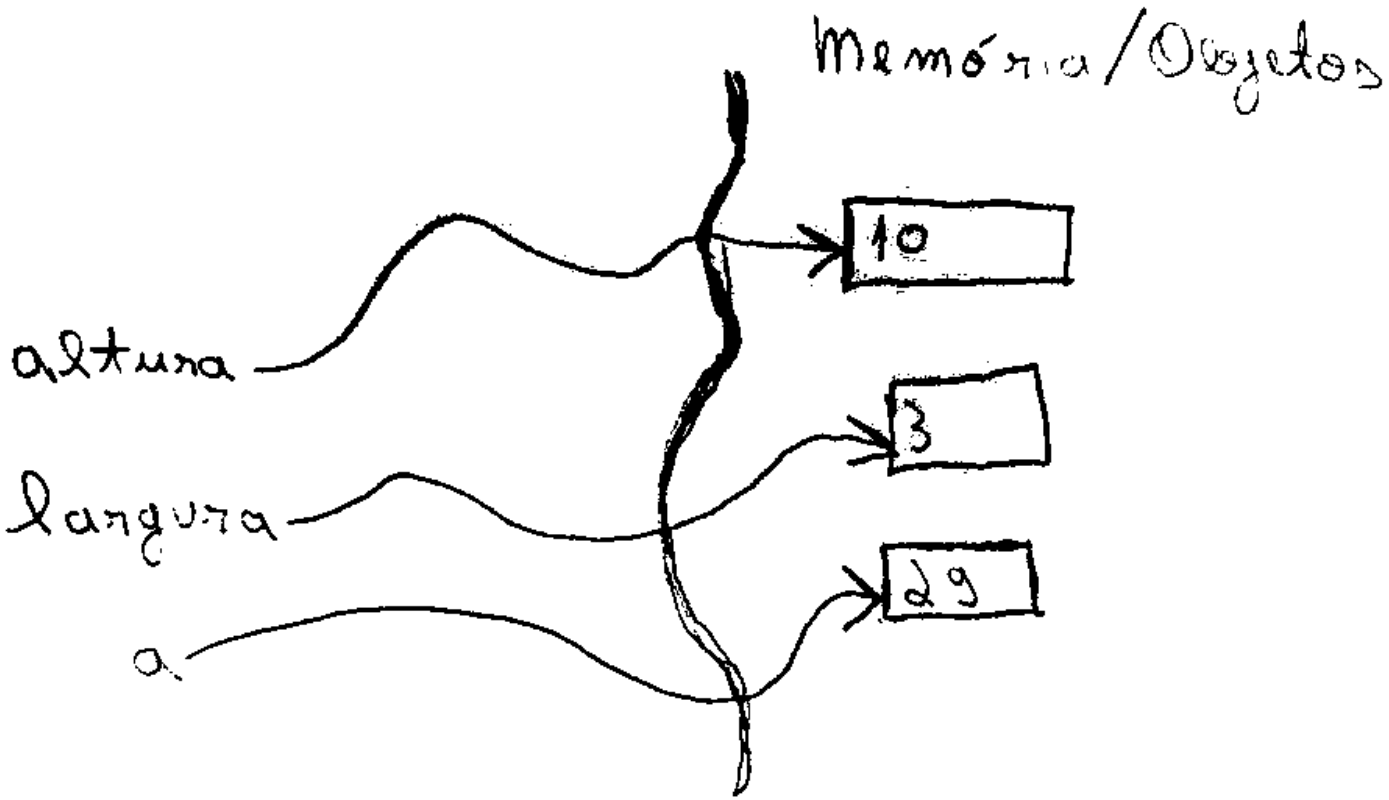
Variáveis são uma forma de se associar um nome dado pelo programador com um objeto.

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

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

Variáveis

```
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:

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

Literais

- Literais são valores que por algum motivo devem aparecer em um programa.
- No programa anterior usamos os literais 10, 3 e 29 que correspondem aos objetos do tipo **int** de Python contendo estes respectivos valores.
- Também usamos anteriormente literais do tipo **string**, como **“Ola turma”**.

Atribuição

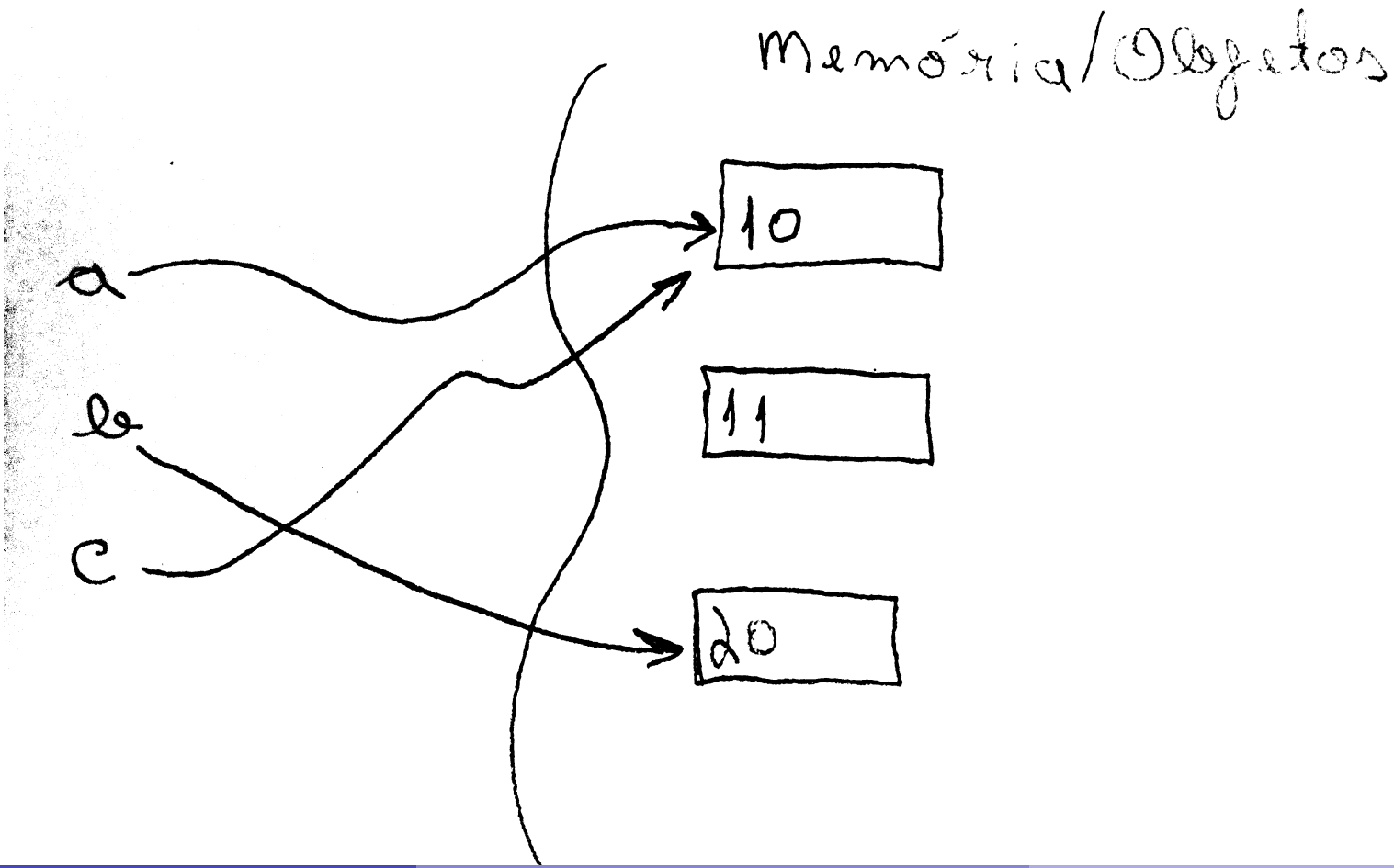
O comando = do Python é o comando de atribuição. Ele associa a variável do lado esquerdo do comando com o objeto do lado direito do comando.

- Um objeto pode ter um nome associado com ele, mais de um nome ou nenhum nome.
- No exemplo abaixo, após todos os comandos serem executados, o objeto 10 terá duas variáveis associadas a ele, o objeto 20 uma, e 11 nenhuma.

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

Atribuição

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



Atribuição

- Se uma variável for usada sem estar associada com nenhum objeto, 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
```

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

Tipos de Objetos 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!".

Os tipos básicos booleano, byte, lista, tupla, conjunto e dicionário serão vistos ao longo 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

- Objetos do tipo **int** armazenam valores inteiros.
- Literais do tipo **int** são escritos comumente como escrevemos 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**.

Tipagem em Python

- Uma variável em Python possui o tipo correspondente ao objeto a que ela está associada naquele instante.
- Python não possui tipagem forte como outras linguagens.
  - Isto significa que você pode atribuir objetos de diferentes tipos para uma mesma variável.
  - Como uma variável não possui tipo pré-definido, dizemos que Python tem **tipagem fraca**.
  - Em outras linguagens criam-se variáveis de tipos específicos e elas só podem armazenar valores daquele tipo para o qual foram criadas.
  - Estas últimas linguagens possuem **tipagem forte**.
- O programa abaixo é perfeitamente legal em Python:

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

- Saída:

```
Olá 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('Olá Pessoal! \n Olá Pessoal')
```

- Saída:

```
Olá Pessoal!
Olá 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 variável contém o valor', a)
```

- Saída:

```
A variável contém o valor 10
```

Escrevendo o conteúdo de uma variável na tela

```
a = 10
b = 3.14
print('a contém o valor', a, '. Já b, contém o valor' , b)
```

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

```
a contém o valor 10. Já b, contém o valor 3.14
```

- Podemos converter todos os valores em strings e usar o operador + para concatenar strings de forma a imprimir sem estes espaços:

```
a = 10
b = 3.14
print('a contém o valor '+str(a)+'. Já b, contém o valor'+str(b))
```

- Saída:

```
a contém o valor 10. Já b, contém o valor3.14
```

Formatos ponto flutuante

- 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("Área do circulo de raio %.2f" %r + " é: %.2f" %area)
print("Área do circulo de raio " + str(r) + " é: " + str(area))
```

- A saída será:

```
Área do circulo de raio 7.00 é: 153.93
Área do circulo de raio 7 é: 153.9335
```

Exemplo

- A função **print** sempre pula uma linha ao final da impressão.
- Se você não quiser que pule uma linha, inclua o argumento **end=' '** no **print**.

```
print("3, ", end="")
print("4, ", end="")
print("5 ", end="")
```

- A saída será:

```
3, 4, 5
```

A função **input**

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

```
print("Digite um número:")
a = input()
print("O número digitado é: " + 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 número digitado vezes 10 é: ", 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 número digitado vezes 10 é %.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.

Exemplo

- O programa abaixo lê dois números e imprime a soma destes.
- Perceba que podemos incluir um texto a ser impresso diretamente no comando **input**.

```
a = float(input("Digite um número:"))
b = float(input("Digite um número:"))
print("A soma dos números é: %.2f" %(a+b))
```

Expressões

- Já vimos que 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**.

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. Se um deles for ponto flutuante faz uma divisão truncada.

```
>>> 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

Mais sobre o operador resto da divisão: %

- Quando computamos "a dividido por b", temos como resultado um valor p e um resto $r < b$ únicos tais que

$$a = p * b + r$$

- Ou seja a pode ser dividido em p partes inteiras de tamanho b, e sobrará um resto $r < b$.

Exemplos:
5%2 tem como resultado o valor 1.
15%3 tem como resultado o valor 0.
1%5 tem como resultado o valor 1.
19%4 tem como resultado o valor 3.

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 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.
- Sempre use 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 de um tipo no outro especificado pela função.
- 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.

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 0o
  Ex: `0o10` ($0x10_8 = 8$)

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