



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

Escola Politécnica - Curso de Engenharia de Software

12413 - ALGORITMOS DE PROGRAMAÇÃO, PROJETOS E COMPUTAÇÃO

ASSUNTO:

- expressões aritméticas, variáveis;
- comando de atribuição;
- comando de saída padrão – *print()* e formatação – *format()*;
- comando *input()*;

Profa. Angela de Mendonça Engelbrecht

angel@puc-campinas.edu.br
profa.angela@gmail.com

Expressões Aritméticas

- Devem ser escritas no **formato linear**. Isto é, não podem ser escritas utilizando mais de uma linha, como no caso da divisão: $\frac{x}{y}$, que deve ser escrita em uma **única linha**: x / y
- Formadas por:
 - operadores: adição (+), subtração (-), multiplicação(*), divisão(/ ou //) , resto da divisão (%) e potência (**);
 - constantes;
 - variáveis;
 - funções com retorno.
- Para a mudança da hierarquia dos operadores, são usados **parênteses ()**. Não são usados, nas expressões, os símbolos [] e { }.

Exemplo:

$$\{ 3 [2 (3 + 8 \div 2) - 3] - \sqrt{100} \} \div (3 - 8)$$

É necessário:

- substituir os símbolos: **{ }** e **[]**, por parênteses **()**;
- usar obrigatoriamente o símbolo ***** para a multiplicação e **/** ou **//** para a divisão;
- para a **raiz quadrada**, usamos uma **função** pronta, chamada: **sqrt** (a); de "**square root**" – biblioteca **math**

Seguem as regras gerais da álgebra, principalmente em relação à hierarquia das operações.

$$(3 * (2 * (3 + 8 / 2) - 3) - \text{sqrt}(100)) / (3 - 8)$$

Funções prontas:

Muitas **funções prontas** estão embutidas no Python padrão e podemos utilizá-las sem adicionar qualquer informação adicional ao programa.

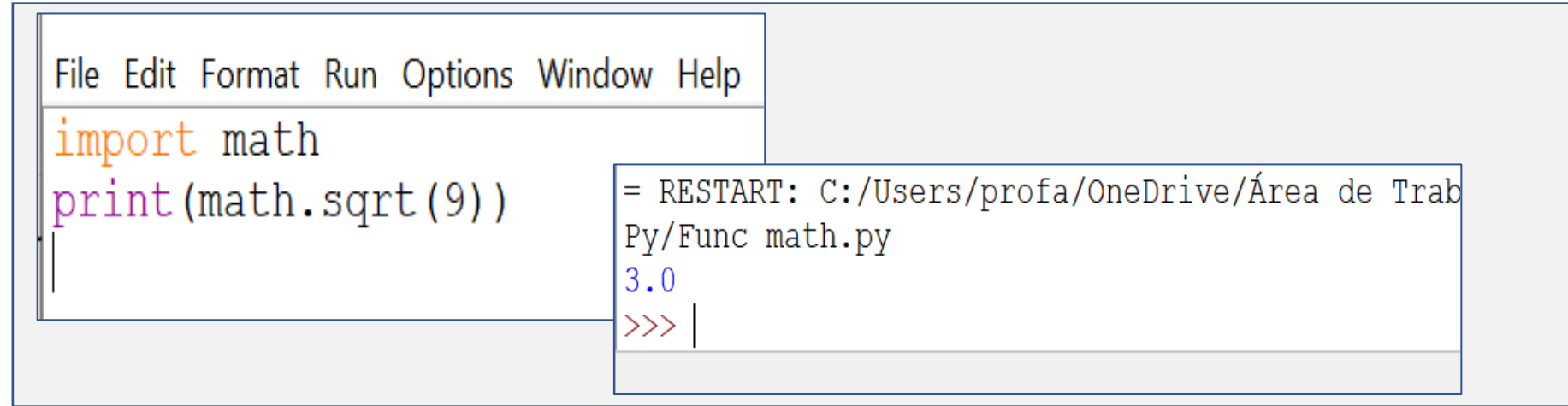
Quando **não** estão **disponíveis**, precisamos **importar** a **Biblioteca específica** para a finalidade que precisamos.

Por **exemplo**, se desejamos calcular a **raiz quadrada** de um número, vamos fazer uso de uma **função** chamada: *sqrt* (x), que pertence à biblioteca *math*.

Para incluir uma biblioteca, usa-se a declaração *import*.

Exemplo: importando funções prontas

Programa:

Biblioteca - *math*

The screenshot shows a Python IDE window with a menu bar (File, Edit, Format, Run, Options, Window, Help) and a code editor. The code in the editor is:

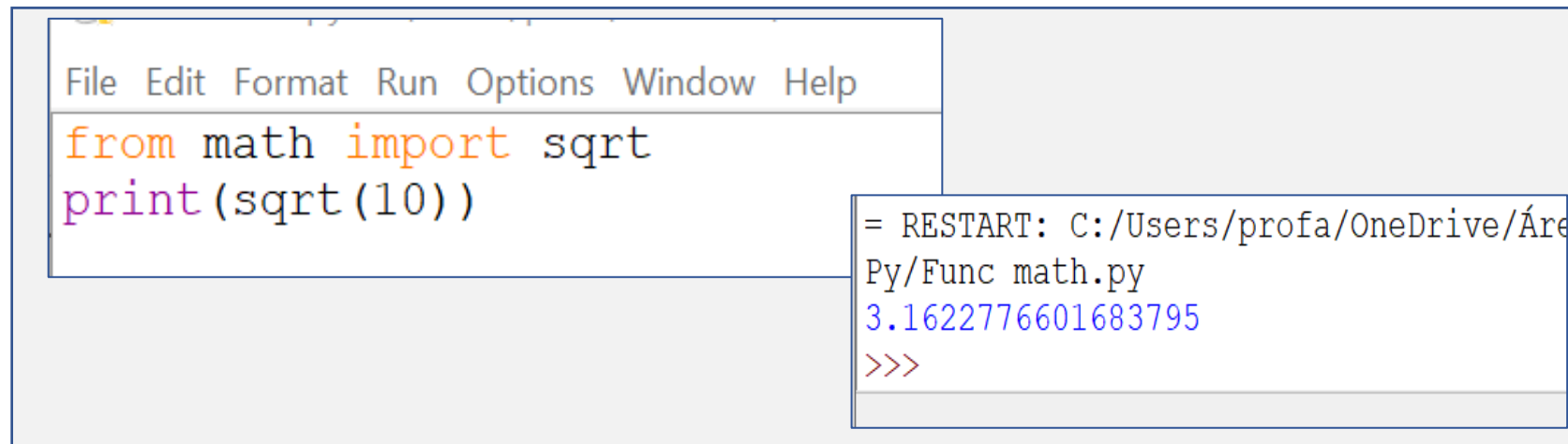
```
import math
print(math.sqrt(9))
```

To the right of the code editor is a console window showing the output of the program:

```
= RESTART: C:/Users/profa/OneDrive/Área de Trab
Py/Func math.py
3.0
>>>
```

ou **importar** apenas a função que deseja:

Programa:



The screenshot shows a Python IDE window with a menu bar (File, Edit, Format, Run, Options, Window, Help) and a code editor. The code in the editor is:

```
from math import sqrt
print(sqrt(10))
```

To the right of the code editor is a console window showing the output of the program:

```
= RESTART: C:/Users/profa/OneDrive/Áre
Py/Func math.py
3.1622776601683795
>>>
```

ou **importar** uma lista:

```
File Edit Format Run Options Window Help
from math import sqrt, sin, cos, pi
print(sqrt(9))
print(cos(pi))
```

```
== RESTART: C:/Users/profa/OneDri
3.0
-1.0
>>>
```

ou **importar** a biblioteca - observe que abre um browser com as funções disponíveis

```
File Edit Format Run Options Window Help
import math
print(math.sqrt(9))
print(math.cos(math.pi))
```

- log10
- log1p
- log2
- modf
- nan
- nextafter
- perm
- pi**
- pow
- prod

```
== RESTART: C:/Users/profa/OneDrive/Á
3.0
-1.0
>>> |
```

Em: <https://docs.python.org/3.6/library/functions.html> lista de função padrão, já integradas, e que não necessitam da importação de outras bibliotecas.

Variáveis são compostas de letras, números e do símbolo de *underline* (_):
 . começam com letras ou *underline*;

Exemplos de nomes de variáveis:

Corretos:

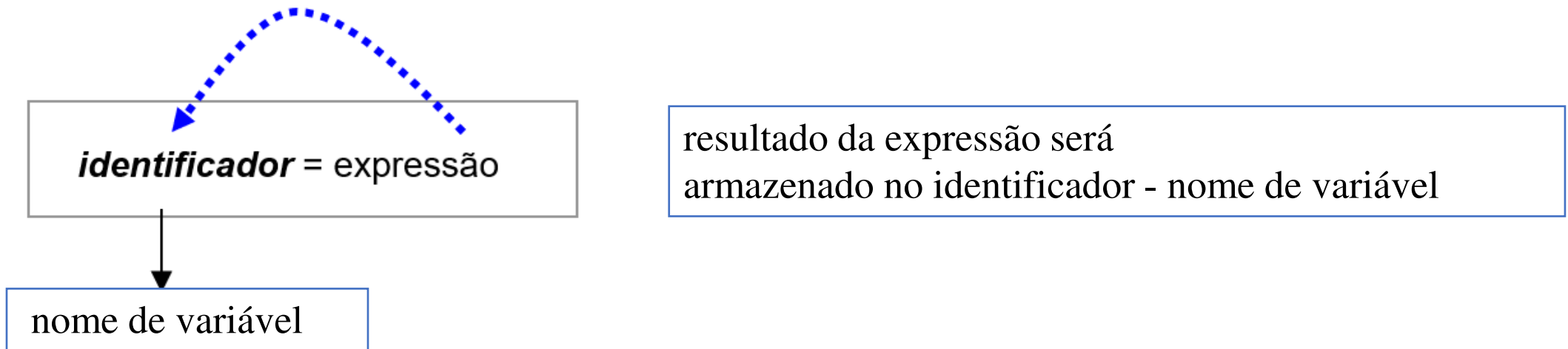
alfa, x, a1, nome_cliente, IdadeAluno, _alfa <- começam com letra, podem ter números e underline (_) também no começo;

beta e Beta <- são diferentes – maiúsculas e minúsculas compõem nomes distintos;

Incorretos:

nota aluno <- não pode haver espaço na composição do nome de uma variável;
 2b <- não pode começar com número;
 if <- **if** é uma palavra reservada da linguagem – comando condicional;


Um **comando**, de modo geral, pode ser definido como uma **ação** a ser executada. A representação abaixo, representa o **comando para a atribuição** de um valor ao identificador (variável) explicitado no lado **esquerdo** do símbolo '='. O valor pode ser o resultado de uma expressão, o valor de uma variável ou uma constante.




análise do comando:


- **identificador**: nome de **uma** variável e deve estar, **OBRIGATORIAMENTE**, no **lado esquerdo**;
- o sinal '=' é utilizado para representar a **atribuição** do resultado da expressão para o identificador;

Não se trata de uma igualdade matemática. É entendido como a **atribuição** de um **valor** para um **identificador**. O identificador por sua vez, não é um número, uma constante. **Exemplos,**

a)  $PI = 3.1416$ significa que à variável **PI** é **atribuído o valor 3.1416**

b)  $A = 5$ → o **valor 5** é **atribuído** à variável **A**

 $B = 3$ → o **valor 3** é **atribuído** à variável **B**

 $A = B$ → considerando-se as atribuições anteriores: à variável **A** é **atribuído o valor 3**, deixando assim, de conter o valor 5.

Para que esta atribuição ($A = B$) não resulte num “erro”, é necessário que a **variável B** tenha valor armazenado, pois é seu valor que é atribuído à variável **A**.

c)  $12 = X$ **ERRO** pois a variável está no lado direito e deveria estar do lado esquerdo

Antes de detalhar a sintaxe da função `print()`, vamos usá-la em alguns **exemplos**, para exemplificar o uso dos **conceitos de variáveis e atribuição, que vimos** - as variáveis e os tipos de dados:

```
File Edit Format Run Options Window Help
# calcular a área de um quadrilátero

base = 4.3
altura = 2.5

area = base * altura

print("Base = ", base)
print("Altura = ", altura)
print("Area = ", area)
```

```
===== RESTART: C:
Base = 4.3
Altura = 2.5
Area = 10.75
>>>
```

Formatação dos dados – há algumas formas de formatarmos os dados - vou apresentá-las de forma gradativa, usando em exercícios e destacando suas definições/sintaxe.

Do exemplo:

sem formatação

```
===== RESTART: C:\
Base = 4.3
Altura = 2.5
Area = 10.75
>>>
```

com formatação – por exemplo
padronizar para 2 casas decimais

```
===== RESTART: C:\
Base = 4.30
Altura = 2.50
Area = 10.75
>>>
```

observe que os valores estão com **2 casas** decimais

se a escolha fosse para **UMA casa decimal**, o valor da base seria arredondado para 10.8

Formatação dos dados – se refere à composição de uma *string*

Python possui basicamente 3 formas para a composição da *string* para impressão dos dados:

- usando o símbolo **%** para a formatação dos dados - obsoleto;
- o método ***.format***;
- usando ***f-string***

observe, no programa, as diferenças:

sem formatação

```
print("Base = ", base)
print("Altura = ", altura)
print("Area = ", area)
```

```
===== RESTART: C:
Base = 4.3
Altura = 2.5
Area = 10.75
>>>
```

com formatação:

usando %

%variável e **retirar** a vírgula

```
print("Base = %.2f" %base)
print("Altura = %.2f" %altura)
print("Area = %.2f" %area)
```

indicativo que há formato a ser processado

```
===== RESTART: C:
Base = 4.30
Altura = 2.50
Area = 10.75
>>>
```

usando **%** - chamado de **especificadores de formato** – **obsoleto** – pouco uso – herança do C – **porém** sua base explica elementos das demais formatações

```
print("Base = %.2f" %base)
print("Altura = %.2f" %altura)
print("Area = %.2f" %area)
```

sintaxe:

% [tamanho] [.precisão] **caractere do tipo**⁽¹⁾

O **caractere** para o formato depende do **tipo** do argumento a ser impresso:

Caractere do tipo ⁽¹⁾	Tipo do Argumento	Descrição
d ou i	int	Para formatar um número inteiro decimal com sinal
f	float	Para formatar números reais com sinal -
s	string	Para formatar uma string

observe, no programa, as diferenças:

com formatação:

usando %

```
print ("Base = {:.2f}" %base)  
print ("Altura = {:.2f}" %altura)  
print ("Area = {:.2f}" %area)
```

%variável e retirar a vírgula

usando método **.format()**

: quando tiver formato

{ } no lugar de % e **.format**(variável)

```
print ("Base = {:.2f}" .format(base) )  
print ("Altura = {:.2f}" .format(altura) )  
print ("Area = {:.2f}" .format(area) )
```

observe, no programa, as diferenças:

com formatação:

usando método *.format()*

```
print("Base = {:.2f}".format(base) )  
print("Altura = {:.2f}".format(altura) )  
print("Area = {:.2f}".format(area))
```

{ } no lugar de % e *.format*(variável)

usando *f-string* – a partir
da versão 3.6

```
print(f"Base = {base:.2f}")  
print(f"Altura = {altura:.2f}")  
print(f"Area = {area:.2f}")
```

f

{variável}

usando método **.format()**

```
print("Base = {:.2f}".format(base) )
print("Altura = {:.2f}".format(altura) )
print("Area = {:.2f}".format(area) )
```

The general form of a *standard format specifier* is:

```
format_spec ::= [[fill]align][sign][#][0][width][grouping_option][.precision][type]
fill        ::= <any character>
align       ::= "<" | ">" | "=" | "^"
sign        ::= "+" | "-" | " "
width       ::= digit+
grouping_option ::= "_" | ","
precision   ::= digit+
type        ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "
```

- as letras **d** e **s** não são necessárias;
- **f** permanece para formatação de **float** – principalmente em situações que se deseja um número de casas decimais específicas. Exemplo: dinheiro: quando os centavos são zeros (.00) e devem ser impressos – R\$ 32.00;
- **alinhamento** à **esquerda** '**<**' e '**>**' para alinhamento à direita e '**^**' para centralizado no espaço especificado – se maior que o tamanho do valor a ser impresso;

```
File Edit Format Run Options Window Help
#exemplos do método .format()
nome = 'Angela Engelbrecht'
idade = 30
altura = 1.50

print(' |Nome:{}|Idade:{}|Altura:{}\n'.format(nome, idade, altura))
print(' |Nome: {:25}|Idade: {:3}|Altura: {:.2}\n'.format(nome, idade, altura))
print(' |Nome: {:<25}|Idade: {:<3}|Altura: {:.2f}\n'.format(nome, idade, altura))
print(' |Nome: {:>25}|Idade: {:<3}|Altura: {:.2f}\n'.format(nome, idade, altura))
print(' |Nome: {:^25}|Idade: {:<3}|Altura: {:.2f}\n'.format(nome, idade, altura))
```

observe a diferença entre os formatos da impressão da altura:

- **nenhum** → o resultado sem formato
- **.2** → até 2 casas
- **.2f** → 2 casas

```
===== RESTART: C:/Users/profa/OneDrive/Área de Trabalho
|Nome:Angela Engelbrecht|Idade:30|Altura:1.5
|Nome:Angela Engelbrecht          |Idade: 30|Altura:1.5
|Nome:Angela Engelbrecht          |Idade:30 |Altura:1.50
|Nome:          Angela Engelbrecht|Idade:30 |Altura:1.50
|Nome:   Angela Engelbrecht      |Idade:30 |Altura:1.50
```

usando ***f-string*** – a partir da versão 3.6 – semelhante ao ***.format()***

```
print(f"Base = {base:.2f}")  
print(f"Altura = {altura:.2f}")  
print(f"Area = {area:.2f}")
```

O que muda:

- **f** ou **F** (de ***formatted***) vai para o início da *string* - antes das aspas → **f"string"** ou **f'string'**
- e, como consequência, as variáveis vão para **dentro** das **chaves**, antes dos dois pontos, quando tiver formato;
- valem as demais especificações de formatos;

File Edit Format Run Options Window Help

```
#exemplos do uso f-string
```

```
nome = 'Angela Engelbrecht'
```

```
idade = 30
```

```
altura = 1.50
```

```
print(f'|Nome:{nome}|Idade:{idade}|Altura:{altura}\n')
```

```
print(f'|Nome:{nome:25}|Idade:{idade:3}|Altura:{altura:.2}\n')
```

```
print(f'|Nome:{nome:<25}|Idade:{idade:<3}|Altura:{altura:.2f}\n')
```

```
print(f'|Nome:{nome:>25}|Idade:{idade:<3}|Altura:{altura:.2f}\n')
```

```
print(f'|Nome:{nome:^25}|Idade:{idade:<3}|Altura:{altura:.2f}\n')
```

A função *input()*: comando para **entrada** de **dados** via **teclado** (entrada **padrão** – *stdin*)

sintaxe:

`input([prompt])`

`[]`: indica opcional

se existir o comando (*prompt*) ele é executado – **ex.**: uma string a ser impressa

Seja o **exemplo**:

File Edit Format Run Options Window Help

leitura de dados - `input()`

`print('<<< Entrada de Dados >>>\n')`

`a = input('Digite algo:')`

`b = input('Digite algo:')`

`c = input('Digite algo:')`

`print('\n<<< Impressão dos Dados Lidos >>>')`

`print('\n a = ', a)`

`print('\n b = ', b)`

`print('\n c = ', c)`

= RESTART: C:/Users/profa/OneDrive/Á
<<< Entrada de Dados >>>

Digite algo:Olá, bom dia!

Digite algo:2021

Digite algo:55.45

<<< Impressão dos Dados Lidos >>>

a = Olá, bom dia!

b = 2021

c = 55.45

>>>

File Edit Format Run Options Window Help

```
#leitura de dados - input()
#exemplo2
```

```
print('<<< Entrada de Dados >>>\n')
a = input('Digite um número inteiro:')
print('Imprimir o número lido adicionado de uma unidade')
print(a + 1)
```

```
===== RESTART: C:/Users/profa/OneDrive/Área de
Trabalho/Programas Py/Leitura input 2.py =====
<<< Entrada de Dados >>>

Digite um número inteiro:1234
Imprimir o número lido adicionado de uma unidade
Traceback (most recent call last):
  File "C:/Users/profa/OneDrive/Área de Trabalho/Programas Py/Leitur
a input 2.py", line 7, in <module>
    print(a + 1)
TypeError: can only concatenate str (not "int") to str
>>> |
```

Observe a mensagem de erro – diz: “**somente pode concatenar *str* com *str* (não “*int*”)**” e, no exemplo, o “**int**” é a constante **1**.

Ele está interpretando o conteúdo de **a** como uma ***string***!

Motivo: as **entradas de dados** são sempre como *string*.

Se queremos fazer a **leitura de números** inteiros e reais então eles precisam ser **convertidos** em números – com *int()* ou *float()*

File Edit Format Run Options Window Help

```
#leitura de dados - input()
#exemplo2
```

```
print('<<< Entrada de Dados >>>\n')
a = int (input('Digite um número inteiro:'))
print('Imprimir o número lido adicionado de uma unidade')
print(a + 1)
|
```

int(): inteiro

float(): real

```
= RESTART: C:/Users/profa/OneDrive/Área de Trabalho/
tura input 2.py
```

```
<<< Entrada de Dados >>>
```

```
Digite um número inteiro:1234
```

```
Imprimir o número lido adicionado de uma unidade
1235
```

```
>>> |
```


Exemplo – usando `type()` para mostrar os tipos assumidos para as variáveis digitadas.

```
# leitura de dados - input()

print('<<< Entrada de Dados >>>\n')
a = input('Digite número inteiro: ')
b = input('Digite número real: ')
c = input('Digite frase: ')
print('\n<<< Impressão dos Dados Lidos >>>')
print('\n a = ', a, " tipo :", type(a))
print('\n b = ', b, " tipo :", type(b))
print('\n c = ', c, " tipo :", type(c))

a = int(input('Digite número inteiro: '))
b = float(input('Digite número real: '))
c = input('Digite frase: ')
print('\n<<< Impressão dos Dados Lidos >>>')
print('\n a = ', a, " tipo :", type(a))
print('\n b = ', b, " tipo :", type(b))
print('\n c = ', c, " tipo :", type(c))
```

```
= RESTART: C:\Users\profa\OneDrive\Áre
<<< Entrada de Dados >>>

Digite número inteiro: 1234
Digite número real: 34.56
Digite frase: Bom dia!

<<< Impressão dos Dados Lidos >>>

a = 1234  tipo : <class 'str'>

b = 34.56  tipo : <class 'str'>

c = Bom dia!  tipo : <class 'str'>
Digite número inteiro: 1234
Digite número real: 34.56
Digite frase: Bom dia!

<<< Impressão dos Dados Lidos >>>

a = 1234  tipo : <class 'int'>

b = 34.56  tipo : <class 'float'>

c = Bom dia!  tipo : <class 'str'>
>>>
```

vamos examinar o exemplo em duas partes:

parte 1:

```
File Edit Format Run Options Window Help
# leitura de dados - input()

print('<<< Entrada de Dados >>>\n')
a = input('Digite número inteiro: ')
b = input('Digite número real: ')
c = input('Digite frase: ')
print('\n<<< Impressão dos Dados Lidos >>>')
print('\n a = ', a, " tipo :", type(a))
print('\n b = ', b, " tipo :", type(b))
print('\n c = ', c, " tipo :", type(c))
```

o três valores lidos são considerados como:
tipo *string* - str

em python, todos os valores lidos são *string* – é necessário converter, se desejar *int* ou *float*

```
= RESTART: C:\Users\profa\OneDrive\Área de Trabalho
<<< Entrada de Dados >>>

Digite número inteiro: 1234
Digite número real: 34.56
Digite frase: Bom dia!

<<< Impressão dos Dados Lidos >>>

a = 1234 tipo : <class 'str'>
b = 34.56 tipo : <class 'str'>
c = Bom dia! tipo : <class 'str'>
```



```
a = int(input('Digite número inteiro: '))
b = float(input('Digite número real: '))
c = input('Digite frase: ')
print('\n<<< Impressão dos Dados Lidos >>>')
print('\n a = ', a, " tipo :", type(a))
print('\n b = ', b, " tipo :", type(b))
print('\n c = ', c, " tipo :", type(c))
```

```
Digite número inteiro: 1234
Digite número real: 34.56
Digite frase: Bom dia!
```

```
<<< Impressão dos Dados Lidos >>>

a = 1234 tipo : <class 'int'>
b = 34.56 tipo : <class 'float'>
c = Bom dia! tipo : <class 'str'>
>>>
```

Exercício: construir um programa que faz a **leitura** de dois valores: um representando a **base** e outro a **altura**.

Calcula a **área do quadrilátero**.

Imprime os **valores lidos** e a **Área** calculada.

Variáveis do problema:

- base e altura – valores lidos: **base e altura -> reais**
- área calculada – valor gerado no programa pela multiplicação dos valores lidos, para a base e altura: **area -> real**