Escola de Engenharia
Algoritmos e Programação I

Profa. Melanie Lerner Grinkraut
melanie.grinkraut@mackenzie.br

Objetivos

- Criar e usar variáveis.
- Identificar e classificar os diversos tipos de dados que serão utilizados em seus programas.
- Realizar operações de entrada e saída de dados
- Conhecer e usar os operadores aritméticos
- Conhecer e usar expressões lógicas

Variáveis & Constantes

- **Qual o valor de y ?**

$$y = x + 2$$

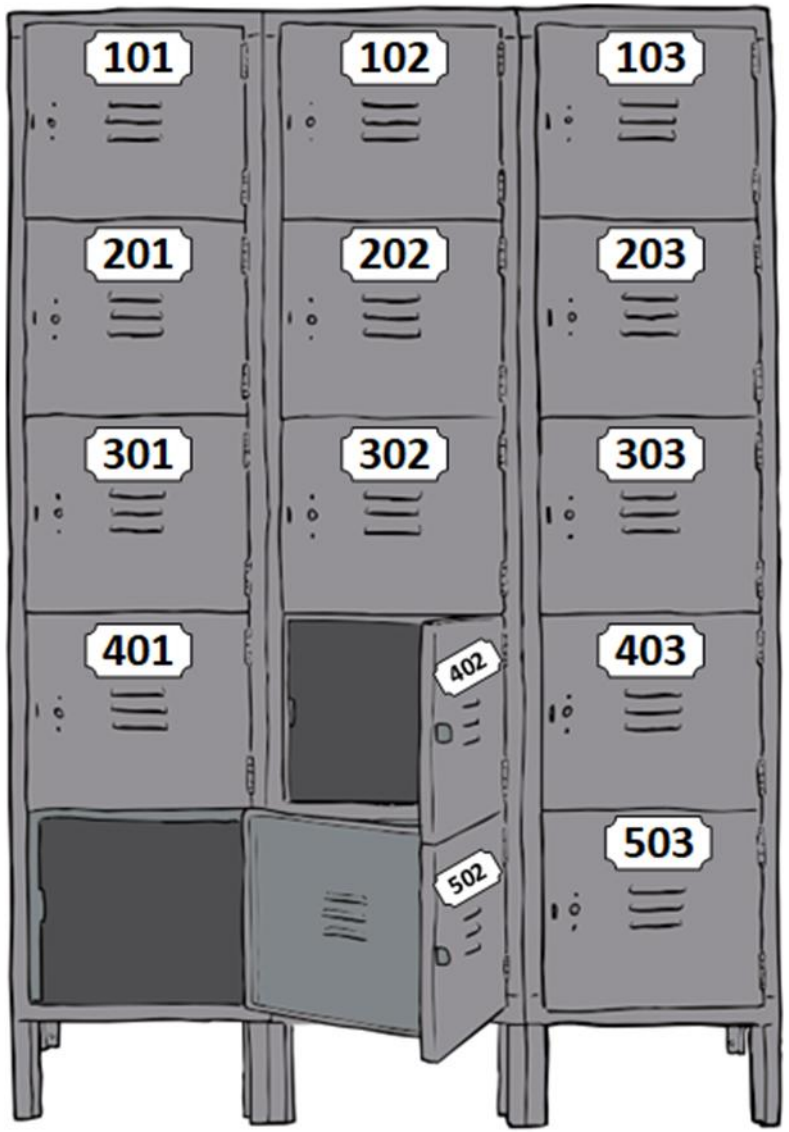
$$\mathbf{x} = 3 \quad \mathbf{y} = 3 + 2 = 5$$

$$\mathbf{x} = 1 \quad \mathbf{y} = 1 + 2 = 3$$

X => Variável

2 => Constante

Variáveis & Constantes



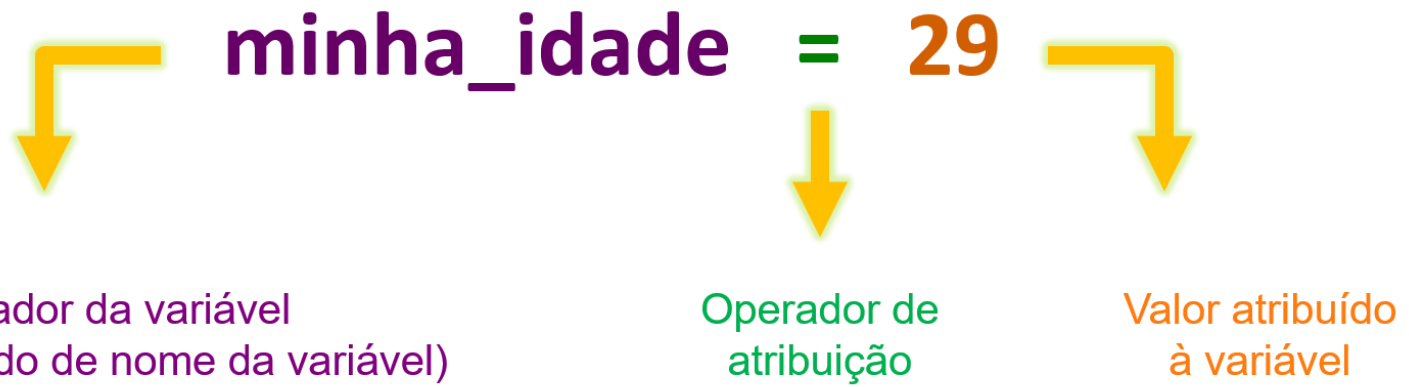
- Muitas vezes, em nossos programas precisamos reservar espaço na memória para guardar valores que o programa acessará e modificará. A este espaço de memória, devidamente rotulado por um nome (**identificador**), chamamos **variável.**
- Quando o espaço de memória possui um valor que não será modificado durante o programa, o chamamos de **constante**.
- Quando queremos guardar um dado na memória, devemos classificá-lo quanto ao seu **tipo**. Pense nos conjuntos matemáticos, tais como inteiros, reais...

Variáveis & Constantes

- Uma **variável** tem um nome (identificador) que está associado a um espaço em memória que armazena um valor. Uma variável pode receber diferentes valores durante a execução de um programa, por isso o nome “variável”;

- Valores são atribuídos às variáveis usando o operador de atribuição (=);

Exemplo:



Variáveis & Constantes

- Um **nome** ou **identificador** de uma variável é formado por uma sequência de um ou mais caracteres. Regras:
  - Pode conter apenas **letras, dígitos e *underscores*** (sublinhas);
  - Pode começar por uma **letra ou *underscore***;
  - Não é permitido o uso de outros **caracteres especiais** (por exemplo, espaço);
  - Não é permitido o uso de **palavras reservadas** da linguagem a ser utilizada;
  - O identificador deve ser conciso, porém **descritivo** (*idade* é melhor que *i*, *tamanho_nome* é melhor que *tamanho_do_nome_da_pessoa*).
  - Python é ***case sensitive***, isto é ele faz distinção entre maiúsculo e minúsculo;

Variáveis & Constantes

Palavras reservadas em Python:

and	as	assert	break	class	continue	def
del	elif	else	except	finally	for	from
global	if	import	in	is	lambda	nonlocal
not	or	pass	raise	return	try	while
with	yield	false	none	true		

Variáveis & Constantes

- **Exercício**

Analise os identificadores e responda o motivo dos identificadores inválidos:

Identificadores válidos	Identificadores inválidos	Motivo
totalVenda	'totalVenda'	
totalvenda	total venda	
venda2016	2016Venda	

Tipos de dados

- **NÚMEROS**
  - Um dado numérico é composto por uma sequência de dígitos (0 a 9), um sinal opcional (+ ou –) e um possível ponto decimal (usa-se o ponto e não a vírgula para separar a parte inteira da fracionária). São classificados como **inteiro** ou de **ponto flutuante**.
    - **int :** números inteiros;
    - **float**: representação de ponto flutuante;

Tipos de dados

- **NÚMEROS**

Exemplos:

Corretos				Incorretos	
Inteiro	Ponto flutuante				
5	5.	5.0	0.125	5,03	
2500	2500.	2500.0	2500.125	2,500	2,500.125
+2500	+2500.	+2500.0	+2500.125	+2,500	+2,500.125
-2500	-2500.	-2500.0	-2500.125	-2,500	-2,500.125

Tipos de dados

- **NÚMEROS**
  - Um dado numérico é composto por uma sequência de dígitos (0 a 9), um sinal opcional (+ ou –) e um possível ponto decimal (usa-se o ponto e não a vírgula para separar a parte inteira da fracionária). São classificados como **inteiro** ou de **ponto flutuante**.
    - **int :** números inteiros;
    - **float**: representação de ponto flutuante;

Tipos de dados

- **STRINGS**
  - Uma string representa uma sequência de caracteres (letras, dígitos, símbolos especiais). Em Python, as strings podem ser delimitadas por um par de aspas simples (’) ou duplas (”).
  - Exemplos:
    - “olá”
    - ‘Universidade Presbiteriana Mackenzie’
    - ‘Rua da Consolação, 930’

Entrada e saída de dados

- Os programas que iremos escrever poderão obter e apresentar dados aos usuários.

- A função **print** (saída de dados), é usada para exibir informações na tela:

```
>>> print('Hello World!')   # 1 argumento.
Hello World!
>>> print(“2 + 2 =“, 2+2)   # 2 argumentos.
2 + 2 = 4
>>> a = 10
>>> print('oi’, a, 'tchau') # 3 argumentos.
oi 10 tchau
```

Comentários

- Os comentários são úteis para auxiliar outros programadores (e a nós mesmos) sobre o que está acontecendo no código e, principalmente, a razão;
- A **linha prefixada com #** será considerada um comentário e ignorada pelo Python;

Exemplo:

```
# Programa feito por: Alex Oliveira
# Data: 16/02/2023
# Versão: 1.0
print('2 + 2 = 4')
```

Demonstração

https://colab.research.google.com/

```
[4]  num = 2
     Num = 3
     NUM = 7.7
     num = 1
     print (Num)
```

Entrada e saída de dados

- Se quisermos pedir ao usuário que forneça um valor, via teclado, usaremos a função **input**. Neste caso, um cursor será exibido, aguardando a digitação do dado (qualquer dado poderá ser digitado!).

- Para que o usuário saiba o que deve ser informado, podemos exibir uma mensagem da seguinte forma:

```
nome = input('Seu nome: ')       # "Megan"
valor = input('Valor: ')         # "2000.0"
parcelas = input('Parcelas: ')   # "5"
```

Entrada e saída de dados

- A função input sempre irá retornar uma **string**, então se queremos trabalhar com números é necessário fazer a conversão explicitamente:

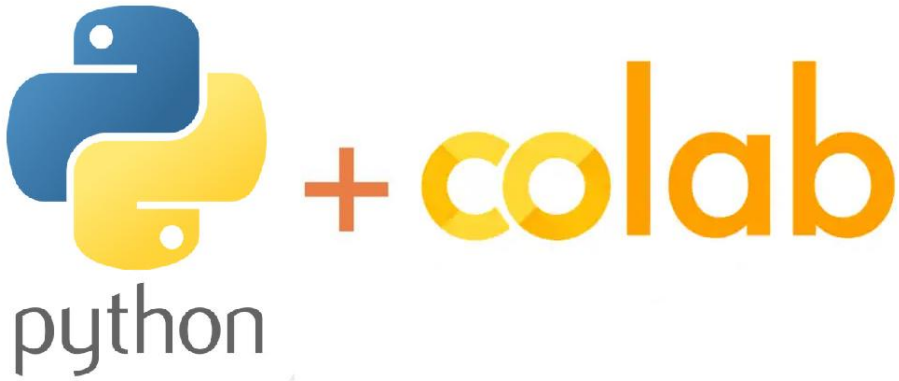
```
valor = float(input('Valor: '))      # 2000.0
parcelas = int(input('Parcelas: '))  # 5
```

Entrada e saída de dados

- Para saber qual é o tipo de dado armazenado em uma variável use a função **type**:

```
>>> type(idade)
<class 'int'>
>>> type(altura)
<class 'float'>
>>> type(nome)
<class 'str'>
```

Demonstração



https://colab.research.google.com/

```
[9]  nome = input('Seu nome: ')
     idade = input('Sua idade: ')
```

Operadores

- Agora que conhecemos os tipos numéricos e string em Python, vamos ver como fazer operações com esses tipos.
- Um **operador** é um símbolo que representa a operação que pode ser realizada em um ou mais operandos. Operadores que atuam sobre um operando são chamados **operadores unários**. Operadores que atuam sobre dois operandos são chamados de **operadores binários**.

Operadores

Operadores	Significado	Exemplo	Resultado
-x	negação	-10	-10
x + y	soma	2 + 4	6
x - y	substração	2 - 4	-2
x * y	multiplicação	2 * 4	8
x / y	divisão	25 / 10	2.5
x // y	divisão truncada	25 // 10	2
		25 // 10.0	2.0
x % y	Módulo ou resto de divisão	25 % 10	5
x ** y	exponenciação	2 ** 4	16

- O operador (-) pode ser um operador unário (negação) ou um operador binário (subtração). Todos os outros operadores são binários.
- Python possui dois operadores para divisão, a divisão dita normal e a divisão truncada, em que o tipo resultante depende dos operandos. Caso os operando sejam inteiros, o resultado da divisão truncada será um inteiro. Caso ao menos um dos operandos seja float, o resultado será um float.

Operadores

- A exponenciação pode ser utilizada tanto para números inteiros quanto para números em ponto flutuante, tanto na base quanto no expoente.
- Exemplos:

```
>>> 2**4
16
>>> 2.5**1.2
3.002811084953578
```

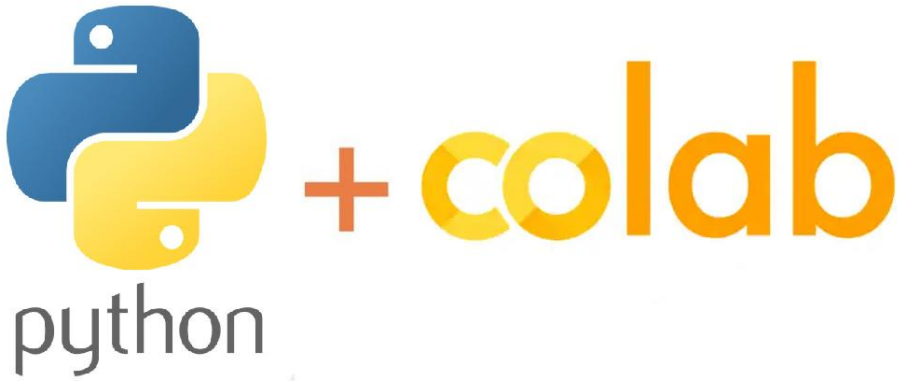
Operadores

- Para números inteiros, Python utiliza um sistema de precisão ilimitada. Já o número de ponto flutuante tem variação e precisão limitada – usa o formato padrão de dupla precisão – fornecendo uma variação de 10^{-308} a 10^{308} com **16 a 17 dígitos de precisão**.

- Para indicar uma variação tão grande de valores, números de ponto flutuante podem ser representados em notação científica:

9.0045602e+5	(9.0045602 x 10^{5}, 8 dígitos de precisão)
1.006249505236801e8	(1.006249505236801 x 10^{8}, 16 dígitos de precisão)
4.239e-16	(4.239 x 10^{-16}, 4 dígitos de precisão)

Demonstração



https://colab.research.google.com/

```
num1 = 10
num2 = 3
num3 = 15
num1 + num3
#num4 = num1/num2
#print (num4)
```

Tipos de Erros

- **Erros de sintaxe:** erros na escrita das instruções da linguagem, violando regras e estruturas;

```
>>> n = 2 + 3) * 5
SyntaxError: unmatched ')'
```

- **Erros em tempo de execução:** erros que só aparecem quando o programa é executado, pois a sintaxe está correta;

```
>>> n = int(input('int: '))
int: ABC
ValueError:  invalid  literal
for int() with base 10: 'ABC'
```

- **Erros de lógica/semântica:** erros relacionados ao algoritmo. Nesses casos, o código-fonte pode ser executado e o programa não gerará nenhuma mensagem de erro, mas o resultado não resolve o problema proposto.

```
>>> print('2 + 5 =', 2 * 5)
2 + 5 = 10
```

Exercício 01

- Desenvolva um programa em Python para determinar o Índice de Massa Corporal (IMC) de uma pessoa. Primeiro elabore o algoritmo, em seguida o fluxograma e depois a codificação;
- Para fazer o cálculo do IMC basta dividir seu peso em quilogramas (Kg) pela altura ao quadrado (em metros);
- O programa deve retornar: “ Olá **Paulo** seu IMC é **25”**

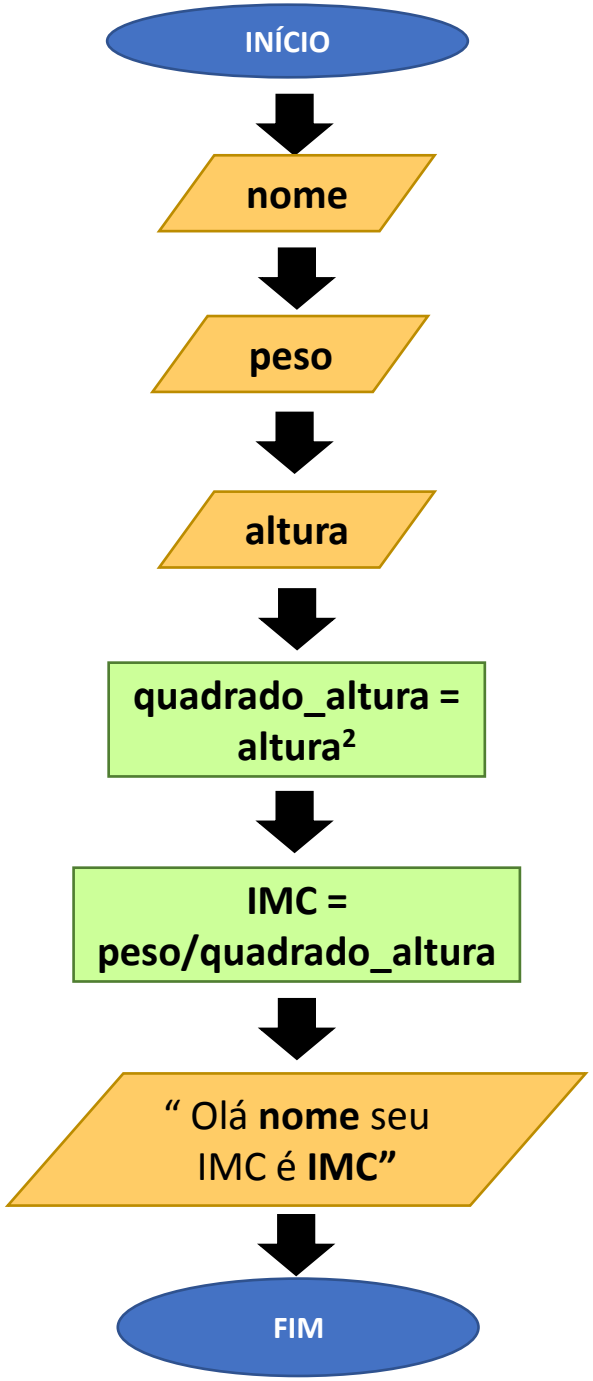
$$IMC = \frac{PESO}{(ALTURA)^2}$$

Exercício 01 - Algoritmo

1. Receber o nome do usuário;
2. Receber o peso do usuário;
3. Receber a altura do usuário;
4. Calcular o quadrado da altura;
5. Calcular o IMC;
6. Exibir nome e IMC;

Exercício 01 – Fluxograma

1. Receber o **nome** do usuário;
2. Receber o **peso** do usuário;
3. Receber a **altura** do usuário;
4. Calcular o **quadrado da altura**;
5. Calcular o **IMC**;
6. Exibir **nome** e **IMC**;



Exercício 01 – Código

1. Receber o **nome** do usuário;
2. Receber o **peso** do usuário;
3. Receber a **altura** do usuário;
4. Calcular o **quadrado da altura**;
5. Calcular o **IMC**;
6. Exibir **nome** e **IMC**;

```
nome = input("Digite seu nome: ")
peso = float(input("Digite seu peso em Kg: "))
altura = float(input("Digite sua altura em metros: "))
quadrado_peso =  altura**2
IMC = peso / quadrado_peso
print("Olá", nome, "seu IMC é",IMC)
```