

## ALGORITMIA E ESTRUTURAS DE DADOS

### MÓDULO II – INTRODUÇÃO AO PYTHON

### TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A WEB

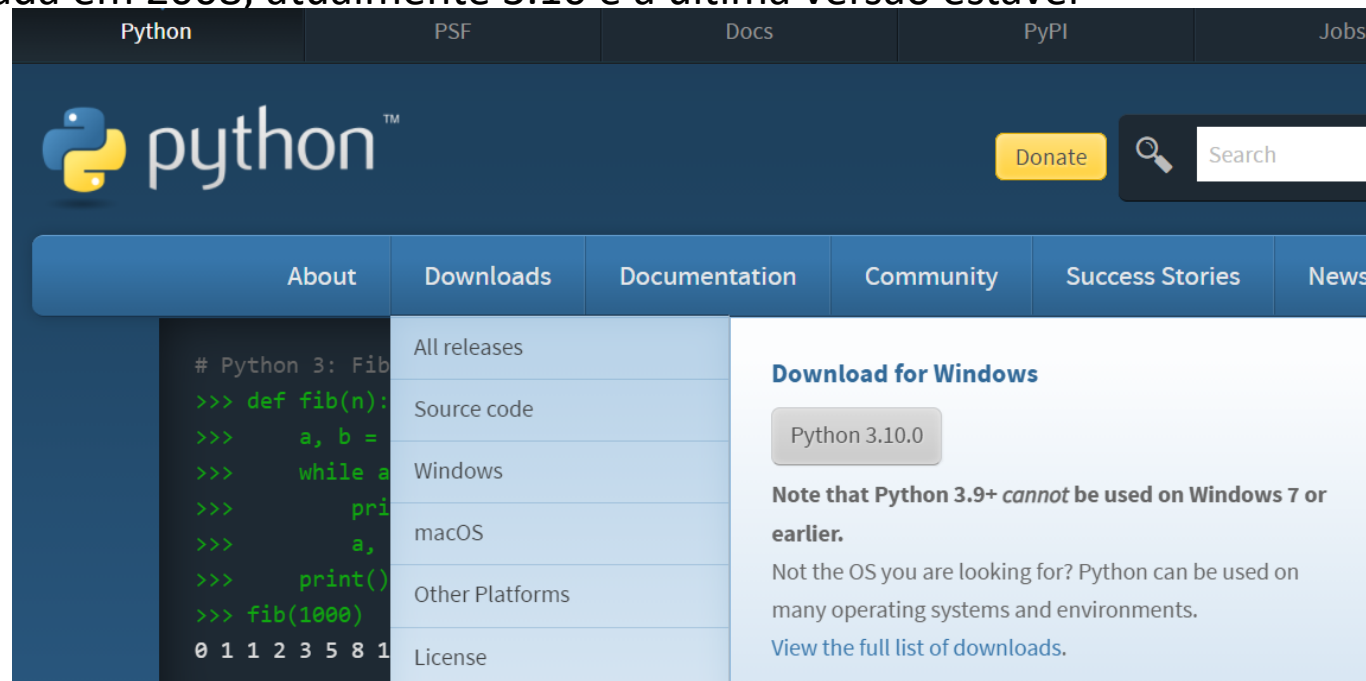
- 1 A Linguagem Python
- 2 Sintaxe Básica e Comentários
- 3 Variáveis
- 4 Tipos de Dados
- 5 Conversões de Dados
- 6 Operadores



# 1

## A Linguagem Python

- ❑ Criada por *Guido Rossum* no início dos anos 90
- ❑ Designação deve-se a série Britânica dos Monthly Python, famosa nos anos 70 e 80 do séc. XX
- ❑ Versão 3.0 lançada em 2008, atualmente 3.10 é a última versão estável



# 1

## A Linguagem Python

- ☐ Linguagem de alto nível
  - ☐ Componente sintática clara, de fácil leitura e interpretação
- ☐ Implementa paradigmas de:
  - ☐ Programação procedimental, estruturada: módulos, funções, estruturas de dados
  - ☐ Programação orientada a objetos
- ☐ Linguagem multiplataforma (Windows, Mac OS, Linux, Raspberry PI)
- ☐ Linguagem open source

# 1

## A Linguagem Python

- ❑ Linguagem de *tipagem dinâmica*
  - ❑ Interpretador do Python infere o tipo dos dados que uma variável recebe, sem necessidade de o explicitar no código
- ❑ Linguagem aplica tipos de dados dinâmicos às variáveis, de acordo com o seu conteúdo num dado momento

```
int peso = int(input ("Peso: "))  
int altura = float(input ("Altura: "))  
float imc= peso / (altura*altura)
```

Exemplo de linguagem de  
*tipagem estática*  
(ex.: C, C++, C#, Pascal)

```
IMC > Ex IMC.py > imc  
1  
2 peso = int(input ("Peso: "))  
3 altura = float(input ("Altura: "))  
4 imc= peso / (altura*altura)  
5 result = "IMC = {}"  
6 print (result.format(imc))
```

Linguagem de tipagem dinâmica  
(ex: python, javascript)

# 1

## A Linguagem Python

- ❑ Linguagem interpretada
  - ❑ Faz uso de um interpretador de código para a execução do programa

Linguagem  
interpretada

- Código fonte transformado em linguagem intermédia, que é interpretada durante a execução do programa
- Código fonte é executado por um interpretador
- Programa gerado não é executado diretamente pelo SO
- Exemplos: Python, Javascript

Linguagem  
compilada

- Processo e conversão do código fonte em linguagem máquina (compilador)
- Geralmente gera aplicações executáveis (.exe), que são executadas pelo SO
- Exemplos: C, C++,

1

## The Zen of Python

### Princípios orientadores da linguagem

Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than \*right\* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!

# 1

## A Linguagem Python

### Princípios orientadores da linguagem

#### ***Beautiful is better than ugly***

*"Bonito é melhor que feio"*

*Se você desenvolve software, sabe que existem vários caminhos para chegar à solução. Em Python, quando is visualmente, é mais elegante! Por exemplo, ao invés de fazer assim:*

```
if funcao(x) && y == 0 || z == 'yes':
```

*prefira:*

```
if funcao(x) and y == 0 or z == 'yes':
```

#### ***Explicit is better than implicit***

*"Explícito é melhor que implícito"*

*Por ser uma linguagem bastante flexível, Python te possibilita solucionar um problema de diversas maneiras partes de código (para que o mesmo fique menor, por exemplo). Em Python nós preferimos a opção mais leg entender (sem nada "escondido"). Por exemplo, use:*

```
import os  
print os.getcwd()
```

*ao invés de:*

```
from os import *  
print getcwd()
```



# 1

## A Linguagem Python

### Princípios orientadores da linguagem

#### *Flat is better than neste*

"Linear é melhor do que aninhado"

Evite criar estruturas dentro de estruturas que estão dentro de outra estrutura (**dicts** são estruturas po aninhá-las: isso resulta em um código mais legível e o acesso ao dado, mais simples. Faça:

```
if i > 0:
    return funcao(i)
elif i == 0:
    return 0
else:
    return 2 * funcao(i)
```

ao invés de

```
if i>0: return funcao(i)
elif i==0: return 0
else: return 2 * funcao(i)
```

#### *Readability counts*

"Legibilidade conta"

Esse tópico é bem simples: ao terminar de desenvolver, olhe seu código passando o olho rapidamente: sobre ele, dando "um tapa no visual"! Um exemplo simples (em Java):

```
public class ClassePrincipal {
    public static void main(String[] args) {
        System.out.println("Olá pythonistas!");
    }
}
```

```
print("Olá pythonistas!")
```

# 1

## A Linguagem Python

### Princípios orientadores da linguagem

***If the implementation is hard to explain, it's a bad idea***

*"Se a implementação é difícil de explicar, é uma má ideia"*

*E novamente a simplicidade é pregada: se você ficou com dúvida sobre a sua própria implementação, revise-a!*

***If the implementation is easy to explain, it may be a good idea***

*"Se a implementação é fácil de explicar, pode ser uma boa ideia"*

*Agora, se a solução é simples de ser explicada, ela pode (repita comigo: **PODE**) ser uma boa ideia. Mas não necessariamente saber explicar é uma boa implementação.*

***Errors should never pass silently. Unless explicitly silenced***

*"Erros nunca devem passar silenciosamente. A menos que sejam explicitamente silenciados"*

*Nunca "silencie" uma exceção, a menos que a mesma seja explicitamente declarada e silenciada. Silenciar uma exceção é um erro grave. Esconder um erro, às vezes inofensivo, às vezes crítico! Portanto, tenha atenção! Não faça isso:*

```
try:
    x = funcao(y)
except:
    pass
```

*Faça, no mínimo:*


```
try:
    x = funcao(y)
except:
    print("Deu ruim!")
```

## 2 Sintaxe básica e comentários

- ❖ *Import*: incorporar módulos ou bibliotecas necessárias ao código
- ❖ Indentação **obrigatória** de blocos de código ( 4 espaços ou um Tab)
- ❖ Mesmo nº de espaços dentro de um nível de indentação


```
import os
import random
# determina se um número é par ou ímpar

number = int(input("Número:"))
# verifica resto da divisão por 2
if number % 2 ==0:
    print("O Número é par")
else:
    print("O número é ímpar")
```



```
code = compile(f.read(), f.name, 'exec')
File "c:\Exercicios Python\EX_ParImpar\ex1.py", line 7
if number % 2 ==0:
^
IndentationError: unexpected indent
PS C:\Exercicios Python>
```

```
EX_ParImpar > ex1.py > ...
1 import os
2 import random
3 # determina se um número é par ou ímpar
4
5 number = int(input("Número:"))
6 # verifica resto da divisão por 2
7 if number % 2 ==0:
8     print("O Número é par")
9 else:
10     print("O número é ímpar")
11
```



## 2 Sintaxe básica e comentários

- ❖ Comentários: iniciam-se com #
- ❖ Comentar diversas linhas """

```
Ex01.py > ...
1  """ Converte polegadas em mm e em cm
2  |     Este comentário pode ter várias linhas
3
4  """
5  pol = int(input("Indique um valor em polegadas:"))
6
7  milimetros= pol*25.4
8
9  # Imprimir resultados
10 print("mm= ", milimetros)
11 print("cm=", milimetros/10)
12
```



### 3

## Variáveis

- ❑ VARIÁVEL: consiste numa estrutura onde são armazenados dados durante a execução de um programa

```
Hello World > Exemplo.py > ...  
1  
2     numero = 25  
3     nome = "Rafael"  
4     print(numero)  
5     print(nome)  
6  
7
```

C:\WINDOWS\py.exe  
25  
Rafael  
\_

### 3

## Variáveis

- ❑ VARIÁVEL: consiste numa estrutura onde são armazenados dados durante a execução de um programa

```
Hello World > Exemplo.py > ...  
1  
2     numero = 25  
3     nome = "Rafael"  
4     print(numero)  
5     print(nome)  
6  
7
```

C:\WINDOWS\py.exe  
25  
Rafael  
\_

- ❑ Variavel = *expressão*

```
Exemplo.py > ...  
1  
2  
3     numero1 = 25  
4     numero2 = 10  
5     media = (numero1 + numero2) / 2  
6     print(media)  
7
```


### 3

## Variáveis


- ❑ REGRAS E CONVENÇÕES DE NOMENCLATURA PARA DEFINIÇÕES DE VARIÁVEIS
  - ❑ O primeiro caracter deve ser uma letra ou um \_
  - ❑ O primeiro caracter não pode ser um dígito
  - ❑ O nome da variável pode consistir em letra (s), número (s) e sublinhado (s) apenas.
  - ❑ Não usar **NUNCA** acentuação
  - ❑ Não incluir espaços
  - ❑ Não incluir caracteres reservados ou ditos especiais (p.e.: .;#&[]-)
  - ❑ Designação deve ser intuitiva

```
ler_array.py  eliminar_dupl.py

Hello World > Exemplo.py > ...
1
2  x = 25
3  y = 10
4  m = (x + y)/2
5  print(m)
6
```



```
Exemplo.py > ...
1
2
3  numero1 = 25
4  numero2 = 10
5  media = (numero1 + numero2) / 2
6  print(media)
7
```

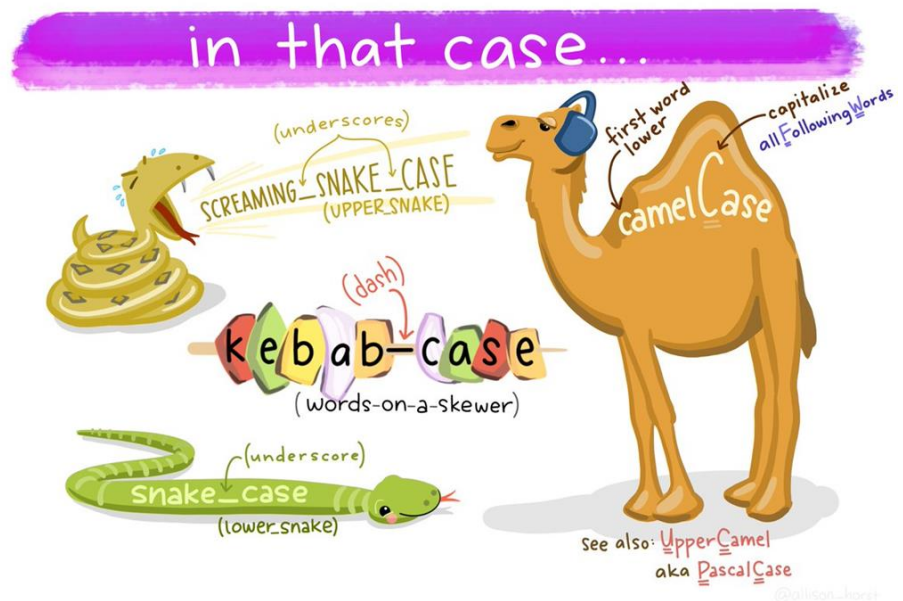




### 3

## Variáveis

- ❑ REGRAS E CONVENÇÕES DE NOMENCLATURA PARA DEFINIÇÕES DE VARIÁVEIS
  - ❑ Não usar nomes demasiado curtos nem demasiado longos
  - ❑ Nomes de variáveis são **case sensitive**
  - ❑ Quando nome de variável inclui 2 ou mais nomes, usar uma das notações:
    - ❑ camelCase
    - ❑ Snake\_case



### 3

## Variáveis

```
Ex02.py > ...  
1  # Converte temperatura de °Celsius para ° Fahrenheit  
2  # forma de conversão: °F = 1.8 * °C + 32  
3  
4  grausCelsius = float(input("° Celsius: "))  
5  grausFahrenheit = 1.8* grausCelsius +32  
6  
7  print("° Fahrenheit: {:.2f} " .format(grausFahrenheit))  
8  
9  input()
```

camelCase

```
# Converte temperatura de °Celsius para ° Fahrenheit  
# forma de conversão: °F = 1.8 * °C + 32  
  
Graus_celsius = float(input("° Celsius: "))  
Graus_fahrenheit = 1.8* Graus_celsius +32  
  
print("° Fahrenheit: {:.2f} " .format(Graus_fahrenheit))  
  
input()
```

snake\_case

### 3

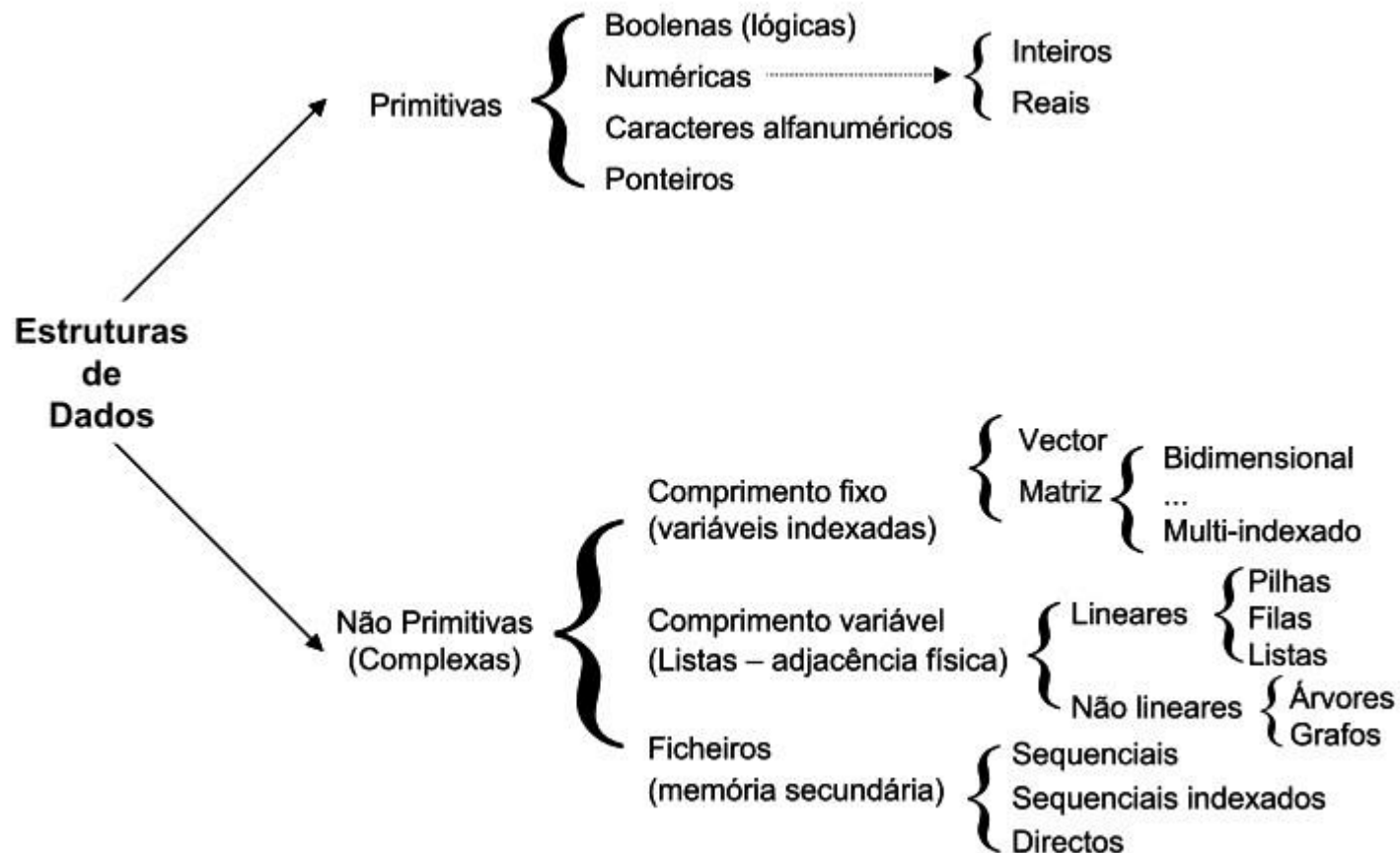
## Variáveis

### ☐ BOAS PRÁTICAS

- ☐ Use nomes legíveis *como* `userName` ou `userPassword`
- ☐ Não use abreviações ou nomes curtos como `a`, `b`, `c`
- ☐ Crie nomes descritivos e concisos. Exemplos de nomes inválidos são `dados` e `valor`. Esses nomes não dizem nada
- ☐ Leve em consideração os termos usados pela equipa de desenvolvimento
- ☐ Não misture línguas naturais (português com inglês)

4

## Estruturas de dados



## 4 Estruturas de dados

Em Python temos vários tipos de dados que vão desde de os tipos numéricos, strings, booleanos, sequencias e coleções.

Tipo de dados	Definição	Descrição
Numéricos	int float	Números inteiros Números reais (vírgula flutuante). Apenas limitados pela memória
Booleanos	bool	Estrutura que pode assumir exclusivamente dois valores: <i>true</i> (1) ou <i>false</i> (0)
Caracteres, Strings	str	Sequência de um ou mais caracteres
Sequências	list, tuplas, range	Listas, tuplos ou range Representam sequências ordenadas de itens
Coleções: Dicionários	dict	Coleções não necessariamente ordenadas de objetos identificados por pares key-value

## 4

## Estruturas de dados

- int : -5, 0, 1000000
- float : -2.0, 3.14159
- bool : True, False
- str : "Hello world!", "K3WL"
- list : [1, 2, 3, 4], ["Hello", "world!"], [1, 2, "Hello"], [ ]

5

## Conversões de Dados (métodos de conversão de dados)

Conversores	Descrição
Int	constrói um número inteiro a partir de um literal inteiro, um literal flutuante (trunca o valor) ou um literal de string (desde que a string represente um número inteiro)
float	constrói um número flutuante a partir de um literal inteiro, um literal flutuante ou um literal de string (desde que a string represente um flutuante ou um inteiro)
str	constrói uma string a partir de uma ampla variedade de tipos de dados, incluindo strings, literais inteiros e literais flutuantes

```
exemplo2.py  eliminar_dupl.py  Ex IMC.py  ex1.py  Exemplo

Hello World > Exemplo.py > ...

1
2
3  numero = 2.82
4  print(numero)           2.82
5  print(int(numero))      2
6  print(float(numero))    2.82
7  print(str(numero))      2.82
8
9
10
```

## 5

## Conversões de Dados

```
ex1.py > ...
1
2  """
3  é equivalente, pois o Python é uma linguagem de
4  tipagem dinâmica: infere automaticamente o tipo de dados
5  """
6  numero= 2
7  numero1 = int(2)
8  print(numero, numero1)
9
10
11  numero = 2.8
12  numero1 = float(2.8)
13  print(numero, numero1)
```



## 5

## Conversões de Dados

```
1  # converte inteiro para string
2  numero=10
3  texto=str(numero)
4  print(texto)
5
6  # converte float para inteiro
7  numero=12.52
8  numero1=int(numero)
9  print(numero1)
10
11 # converte inteiro para float
12 numero=20
13 numero1=float(numero)
14 print(numero1)
```

c:\Users\mario\OneDrive\AED\4 - Exercicios\Ficha 01 - VS Code Consol

```
10
12
20.0
Press any key to continue . . .
```

## 5

## Conversões de Dados

- ❑ A função **type()** retorna o tipo do dados passado como parâmetro.

```
Hello World > Exemplo.py > ...
1
2
3  numero = 2
4  print (numero)
5  print (type(numero))
6
7  numero = 2.82
8  print (type(numero))
9
10 numero = "2.82"
11 print(str(numero))
12 print (type(numero))
13
```

C:\WINDOWS\py.exe

```
2
<class 'int'>
2.82
<class 'float'>
2.82
<class 'str'>
```

# 6

## Operadores

Categoria	Operadores
Aritméticos	<ul style="list-style-type: none"><li>+ (soma)</li><li>- (subtração)</li><li>* (multiplicação)</li><li>/ (divisão)</li><li>% (resto da divisão)</li><li>** (exponenciação)</li><li>pow (exponenciação)</li><li>// (divisão truncada)</li><li>abs (valor absoluto)</li></ul>

```
numero = 20
numero = numero + 10    # soma
numero = numero / 2
numero = numero % 2     # resto da divisão
numero **2              # esponenciação: numero ao quadrado
pow(numero, 2)          # exponenciação: base e expoente

numero = -10
abs(numero)             # valor absoluto (10)
```

## 6

## Operadores

Algumas formas abreviadas de sintaxe de operações aritméticas:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

## 6 Operadores

```
# determina se um número é par ou ímpar
number = int(input("Número:"))
# verifica resto da divisão por 2
if number % 2 == 0:
    print("O Número é par")
else:
    print("O número é ímpar")
```

```
3 numero1 = int(input("Numero:"))
4 numero2 = int(input("Numero:"))
5 numero3 = int(input("Numero:"))
6
7 if numero1 > numero2 and numero1 > numero3:
8     print("o maior é", numero1)
9 elif numero2 > numero1 and numero2 > numero3:
10    print("o maior é", numero2)
11 else:
12    print("o maior é", numero3)
13
```

Categoria	Operadores
Lógicos	and or not
Relacionais	== != < > <= >= is is not

```
if numero1 == numero2:
    print("os numeros são iguais")
```

```
if numero1 is numero2:
    print("os numeros são iguais")
```

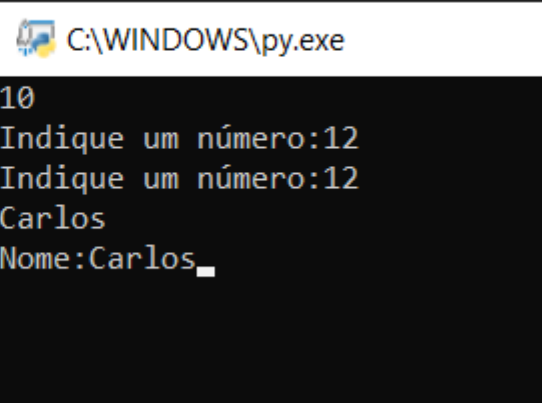
## 6

## Entrada e Saída de Dados

### ❑ Input(*text*)

*text* - string, representa mensagem apresentada antes da entrada de dados

```
1
2
3  numero1 = input()
4  numero1 = input("Indique um número:")
5  numero1 = int(input("Indique um número:"))
6
7  nome = input()
8  nome = input("Nome:")
9
10
```



C:\WINDOWS\py.exe

10

Indique um número:12

Indique um número:12

Carlos

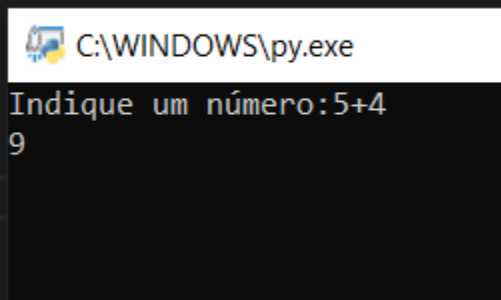
Nome:Carlos\_

## 6

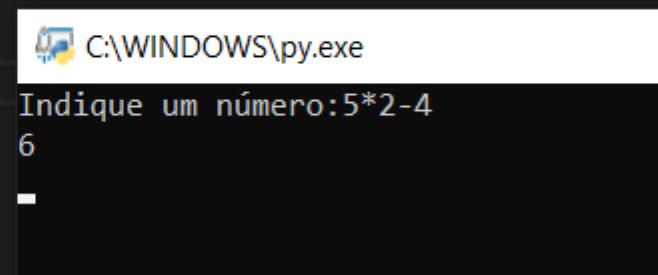
## Entrada e Saída de Dados

- ❑ `Input(text)`  
*eval (expression)* – avalia uma expressão

```
1
2
3  numero = eval(input("Indique um número:"))
4
5  print(numero)
6
7
8
```



```
2
3  numero = eval(input("Indique um número:"))
4
5  print(numero)
6
7
8
9
0
```

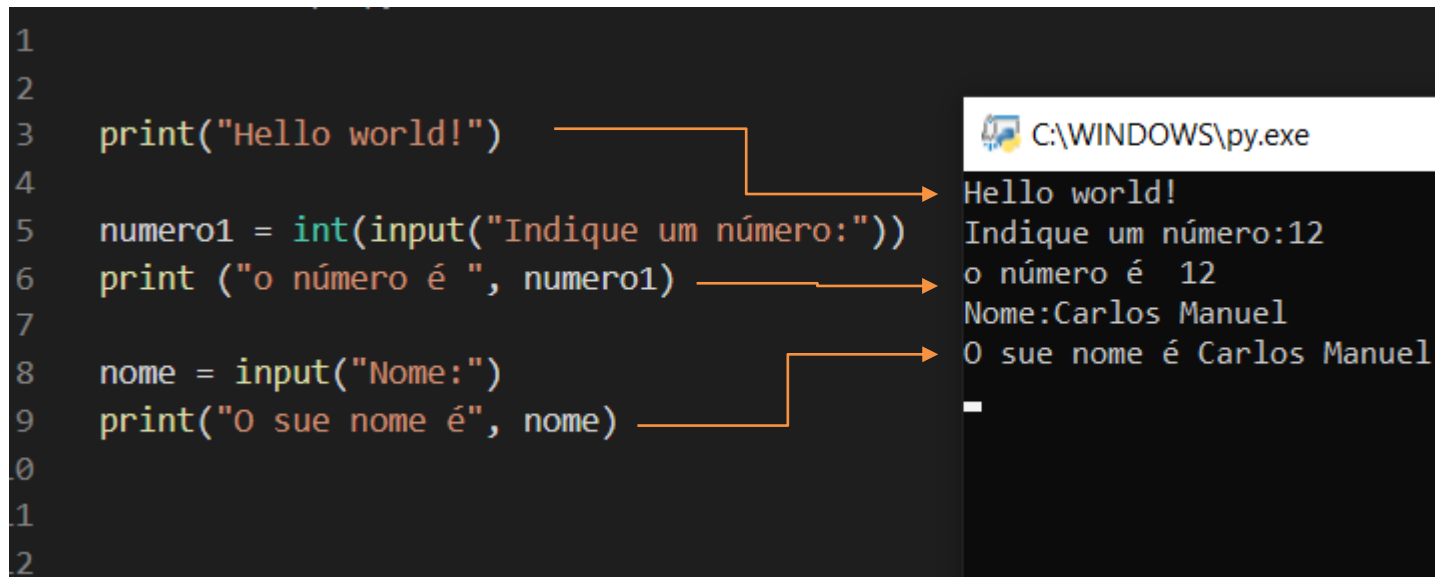


## 6

## Entrada e Saída de Dados

❑ `print(texto, variaveis)`

```
1
2
3  print("Hello world!")
4
5  numero1 = int(input("Indique um número:"))
6  print ("o número é ", numero1)
7
8  nome = input("Nome:")
9  print("O sue nome é", nome)
10
11
12
```



C:\WINDOWS\py.exe

Hello world!  
Indique um número:12  
o número é 12  
Nome:Carlos Manuel  
O sue nome é Carlos Manuel

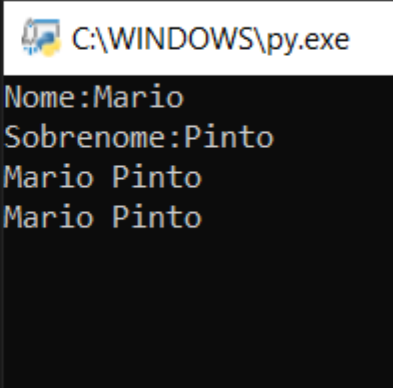


## 6

## Entrada e Saída de Dados

- ❑ `print(texto, variaveis)` - formatando strings (texto)

```
2
3 first_name = input("Nome:")
4 last_name = input("Sobrenome:")
5
6 print(first_name + " " + last_name)
7 # alternativa:
8 print(first_name, last_name)
```

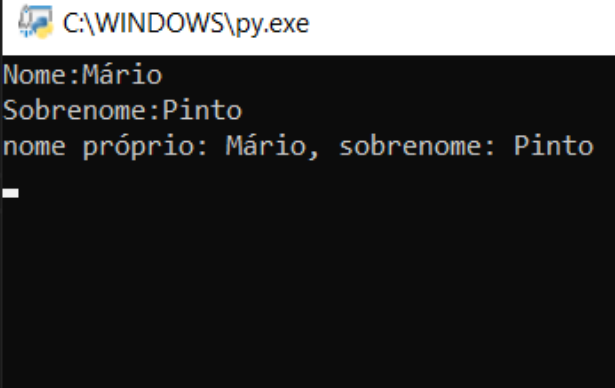


C:\WINDOWS\py.exe

Nome:Mario  
Sobrenome:Pinto  
Mario Pinto  
Mario Pinto

```
first_name = input("Nome:")
last_name = input("Sobrenome:")

print("nome próprio: %s, sobrenome: %s" % (first_name, last_name))
```



C:\WINDOWS\py.exe

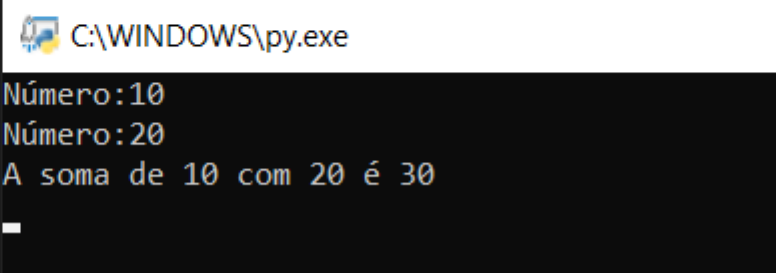
Nome:Mário  
Sobrenome:Pinto  
nome próprio: Mário, sobrenome: Pinto

## 6

## Entrada e Saída de Dados

- ❑ `print(texto, variaveis)` - formatando números

```
2
3 numero1 = int(input("Número:"))
4 numero2 = int(input("Número:"))
5 print("A soma de %s com %s é %s" %(numero1, numero2, numero1+numero2))
6
7
8
9
10
11
12
```

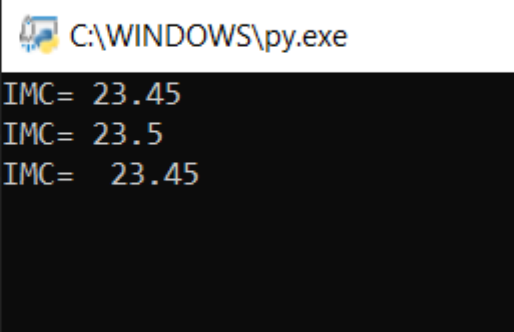


C:\WINDOWS\py.exe

Número:10  
Número:20  
A soma de 10 com 20 é 30

```
numero = 23.453

print ("IMC= %.2f" %numero)
print ("IMC= %.1f" %numero)
print ("IMC= %.6.2f" %numero)
```



C:\WINDOWS\py.exe

IMC= 23.45  
IMC= 23.5  
IMC= 23.45

## 6

## Entrada e Saída de Dados

- ❑ `print(texto, variaveis)` - o método `format()`

```
imc = 21.721  
print("IMC = {:.2f}" .format(imc))  
print("IMC = {:.1f}" .format(imc))  
print("IMC = {:.6.2f}" .format(imc))
```

C:\WINDOWS\py.exe

```
IMC = 21.72  
IMC = 21.7  
IMC = 21.72  
_
```

6 caracteres

float

2 casas decimais

## 6

## Entrada e Saída de Dados

❑ `print(texto, variaveis)` - o método *format()*

```
horas = 4
minutos = 10
tempo = 25

print("{0} horas, {1} minutos, {2} segundos".format(horas, minutos, tempo) )

print(horas, "horas,", minutos, "minutos,", tempo, "segundos")
```

C:\WINDOWS\py.exe

```
4 horas, 10 minutos, 25 segundos
4 horas, 10 minutos, 25 segundos
```