

POLYTECHNIC
FROM PORTO
SCHOOL
HIGHER
MEDIA ARTS
AND DESIGN

P.PORTO

ALGORITHMY AND DATA STRUCTURES

MODULE II – INTRODUCTION TO PYTHON

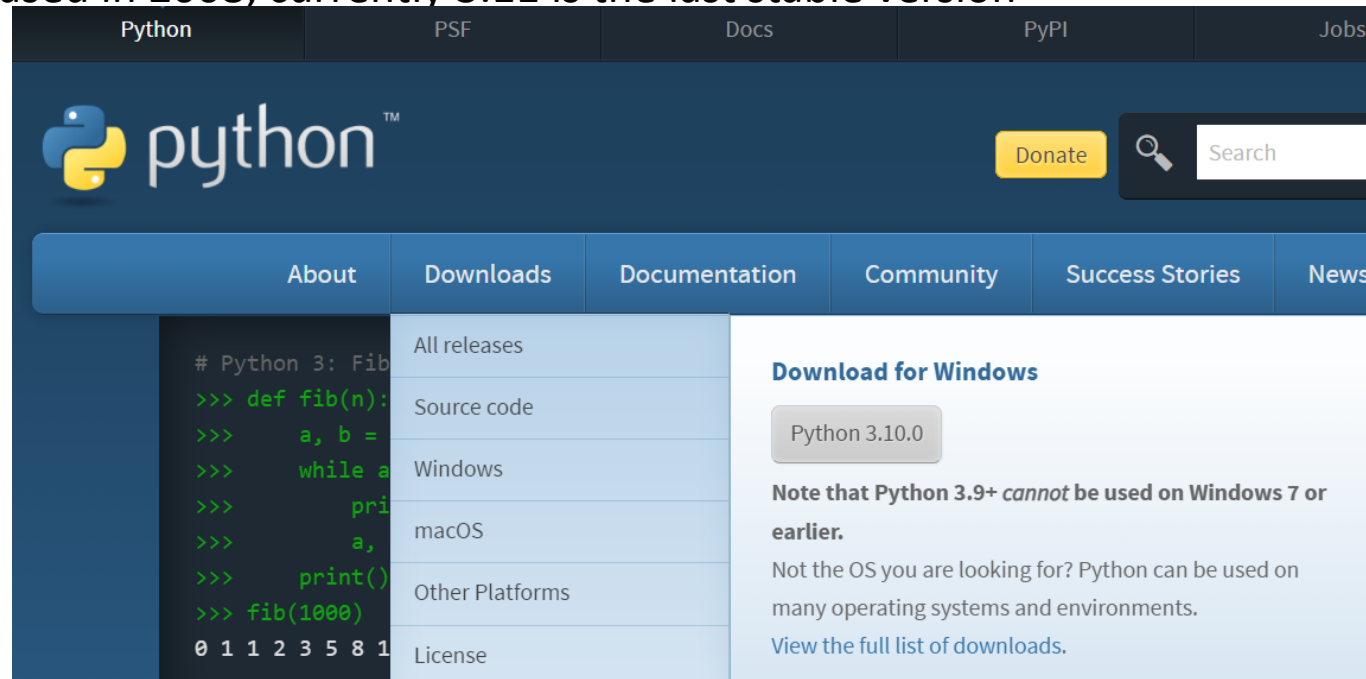
TECHNOLOGIES AND INFORMATION SYSTEMS FOR THE WEB

- 1 The Python language
- 2 Basic Syntax and Comments
- 3 Variables
- 4 Data Types
- 5 Data Conversions
- 6 Operators



1 The Python language

- ❑ Created by *Guido Rossum* in the early 90s
- ❑ Named after the British series of Monthly Python, famous in the 70s and 80s of the century. XX
- ❑ Version 3.0 released in 2008, currently 3.11 is the last stable version



1 The language Python

- ☐ High-level language
 - ☐ Clear syntactic component, easy to read and interpret
- ☐ Implements paradigms of:
 - ☐ Procedural, structured programming: modules, functions, data structures
 - ☐ Object-oriented programming
- ☐ Multiplatform language (Windows, Mac OS, Linux, RaspberryPI)
- ☐ Open source language

1 The language Python

- ❑ Language of dynamic *typing*
 - ❑ Interpreter of Python infers the type of data that a variable receives, without the need to make it explicit in the code
 - ❑ Language applies dynamic data types to variables, according to their content at a given time

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

Example language with static *typing*
(e.g.: 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))
```

Example of language with
dynamic typing
(e.g. python,javascript)

1 The language Python

- ❑ Interpreted language
 - ❑ Makes use of a code interpreter to execute the program

Language
interpreted

- Source code transformed into intermediate language, which is interpreted during program execution
- Source code is executed by an interpreter
- Generated program is not executed directly by SO
- Examples: Python, JavaScript

Language compiled

- Source code process and conversion in machine language (compiler)
- Generally generates executable applications (.exe), which are executed by the OS
- Examples: C, C++,

1

The Zen of Python

Language Guiding Principles

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 The language Python

Language Guiding Principles

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 The language Python

Language Guiding Principles

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 The language Python

Language Guiding Principles

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 Basic syntax and comments

- ❖ *Import*: embed necessary modules or libraries into the code
- ❖ Indentation **mandatory** of code blocks (4 spaces or one Tab)
- ❖ Same number of spaces within an indentation level


```
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 Basic syntax and comments

- ❖ Comments: start with #
- ❖ Comment several lines `"""`

```
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

Variables

- ❑ VARIABLE: consists of a structure where data is stored during the execution of a program

```
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

Variables

- ❑ VARIABLE: consists of a structure where data is stored during the execution of a program

```
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

- ❑ Variable=*expression*

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

3


Variables

❑ NAMING RULES AND CONVENTIONS FOR VARIABLE DEFINITIONS


- ❑ The first character must be a letter or _
- ❑ The first character cannot be a digit
- ❑ The variable name can consist of letter(s), number(s), and underscore(s) only.
- ❑ Do not use **NEVER** special accentuation
- ❑ Do not include spaces
- ❑ Do not include reserved characters or special characters (eg: .;#&[]-)
- ❑ Designation must be intuitive

```
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
```



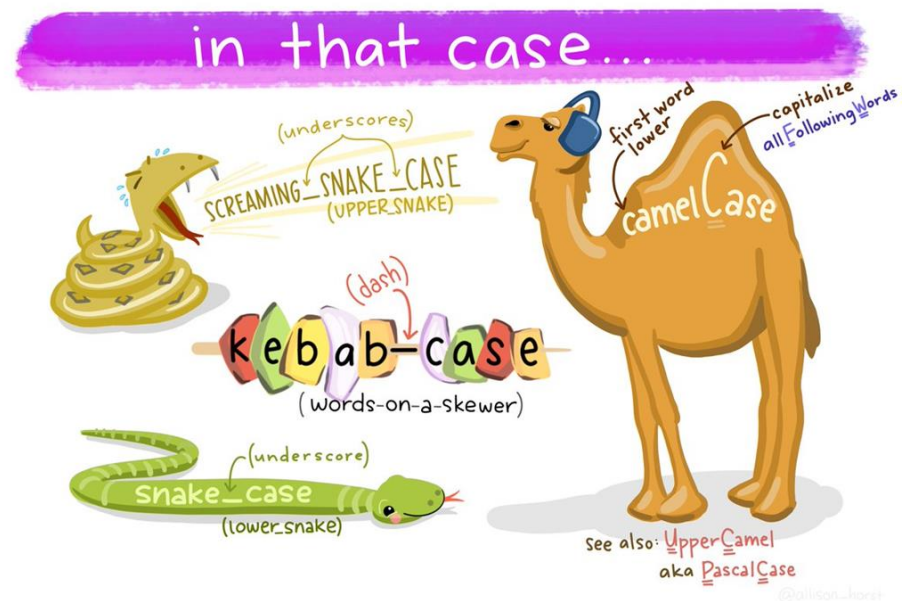
```
Exemplo.py > ...
1
2
3  numero1 = 25
4  numero2 = 10
5  media = (numero1 + numero2) / 2
6  print(media)
7
```



3

Variables

- ❑ NAMING RULES AND CONVENTIONS FOR VARIABLE DEFINITIONS
 - ❑ Do not use names that are too short or too long
 - ❑ Variable names are **case sensitive**
 - ❑ When variable name includes 2 or more names, use one of the notations:
 - ❑ camelCase
 - ❑ Snake_case



3

Variables

```
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

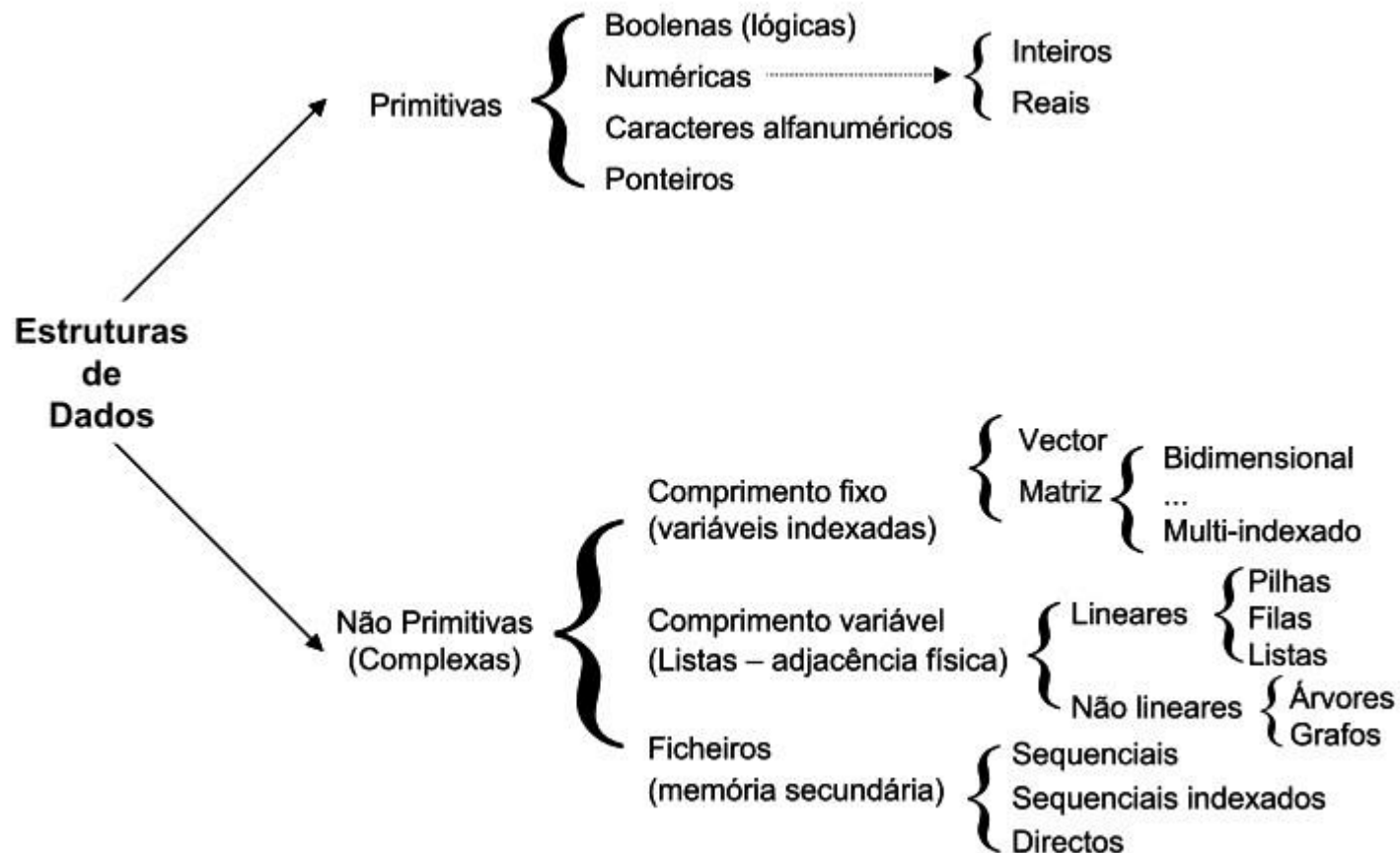
Variables

☐ GOOD HABITS

- ☐ Use readable names *as userName* or *userPassword*
- ☐ Do not use abbreviations or short names such as a, b, c
- ☐ Create descriptive and concise names. Examples of invalid names are data and value. These names don't say anything
- ☐ Take into consideration the terms used by the development team
- ☐ Do not mix natural languages (Portuguese with English)

4

Data structures



4 Data structures

In Python we have several types of data ranging from numeric types, strings, booleans, sequences and collections.

Data type	Definition	Description
Numeric	int float	Whole numbers Real numbers (floating point). Only limited by memory
Booleans	bool	Structure that can exclusively assume two values: <i>true</i> (1) or <i>false</i> (0)
Characters, Strings	str	Sequence of one or more characters
Sequences	list, tuples, range	Lists, tuples Represent ordered sequences of items
Collections: Dictionaries	dict	Not necessarily ordered collections of peer-identified objects key-value

4

Data structures

- 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

Data Conversions (data conversion methods)

Converters	Description
Int	constructs an integer from an integer literal, a float literal (truncates the value), or a float literal string(as long as the string represents an integer)
float	constructs a float number from an integer literal, a float literal, or a string(as long as the string represents a float or an integer)
str	build a string from a wide variety of data types, including strings, integer literals and float literals

```
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
```

The screenshot shows a Python IDE with a file named 'Exemplo.py' open. The code in the file is as follows:

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

The output of the script is shown on the right side of the IDE, with arrows indicating the mapping from the code to the output:

- Line 4: `print (numero)` outputs `2.82`
- Line 5: `print(int(numero))` outputs `2`
- Line 6: `print(float(numero))` outputs `2.82`
- Line 7: `print(str(numero))` outputs `2.82`

The output window title is `C:\WINDOWS\py.exe`.

5

Data Conversions

```
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

Data Conversions

```
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

Data Conversions

- ❑ The function **type()** returns the data type passed as a parameter.

```
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

Operators

Category	Operators
Arithmetic	<ul style="list-style-type: none">+ (sum)- (subtraction)* (multiplication)/ (division)% (rest from the division)** (exponentiation)pow (exponentiation)// (truncated division)abs (absolute value)

```
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 Operators

Some short forms arithmetic operations syntax:

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 Operators

```
# 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	Operators
Logicians	and
	or
	not
Relational	==
	!=
	<
	>
	<=
	>=
	is
	is not

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

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

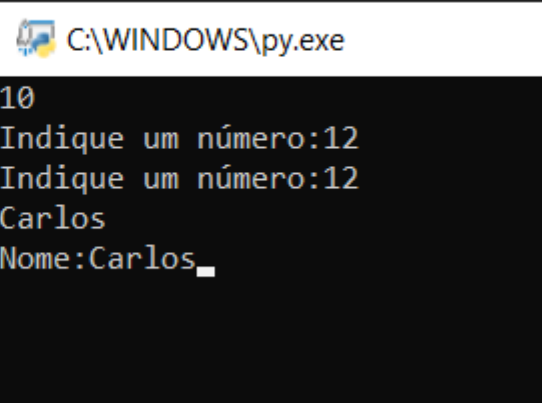
6

Data Input and Output

❑ Input(*text*)

text-string, represents the message presented before data entry

```
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_

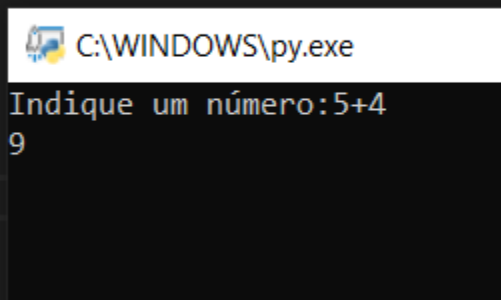
6

Data Input and Output

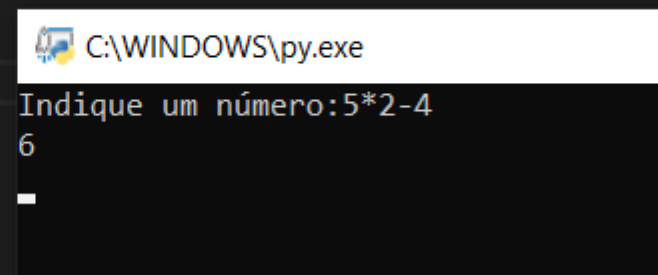
❑ `Input(text)`

eval(expression) –evaluates an expression

```
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
```

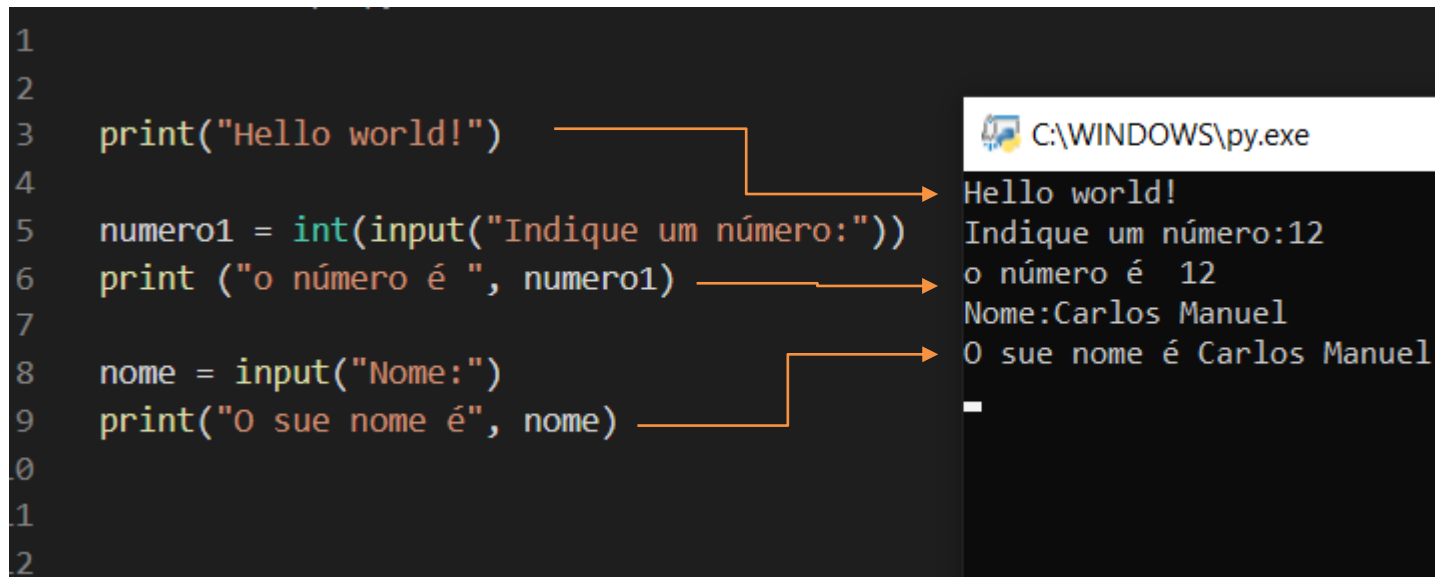


6

Data Input and Output

❏ `print(text,variables)`

```
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

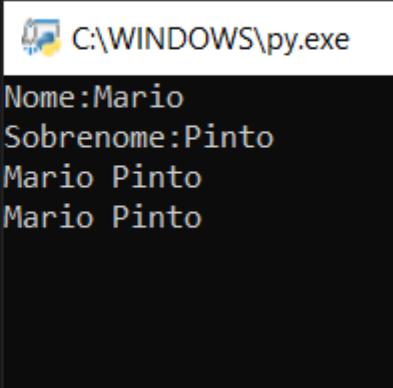
Nome:Carlos Manuel

O sue nome é Carlos Manuel

6 Data Input and Output

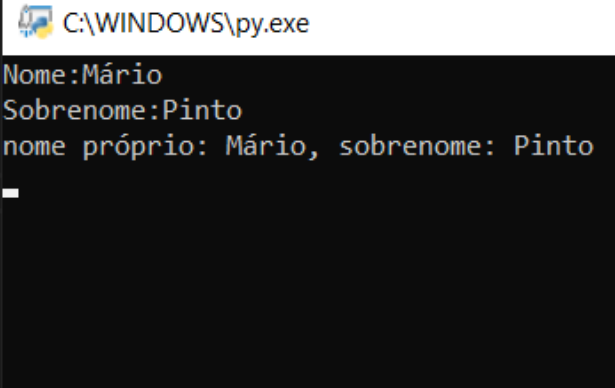
- ❑ `print(text, variables)` –formatting strings (text)

```
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)
```



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

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

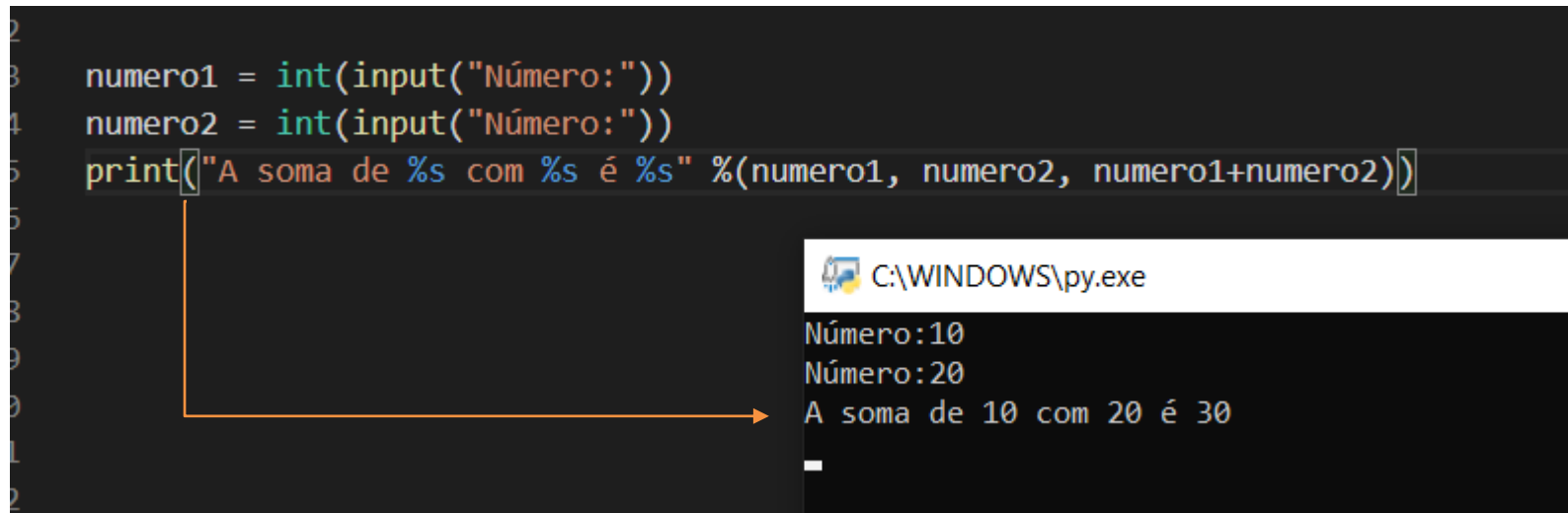


6

Data Input and Output

- ❑ `print(text, variables)` -formatting numbers

```
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
```

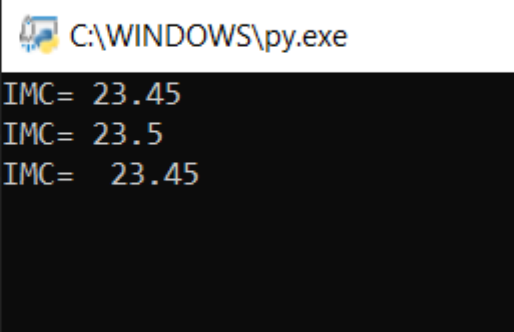


C:\WINDOWS\py.exe

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= %.6f" %numero)
```



C:\WINDOWS\py.exe

IMC= 23.45
IMC= 23.5
IMC= 23.453000

6

Data Input and Output

- ❑ `print(text, variables)` – The `format()` method

```
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 characters

float

2 decimals digits

6

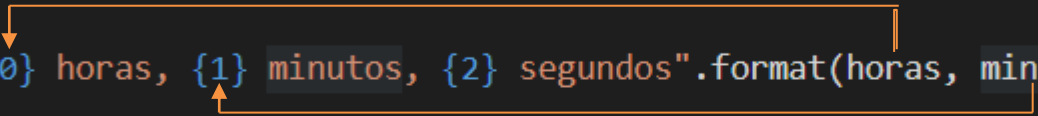
Data Input and Output

❑ `print(text, variables)` - The `format()` method

```
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
```