



# python

**ALGORITMIA E ESTRUTURAS DE DADOS**

**INTRODUÇÃO À LINGUAGEM PYTHON**

**LICENCIATURA EM  
TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A WEB  
#ESMAD #P.PORTO**

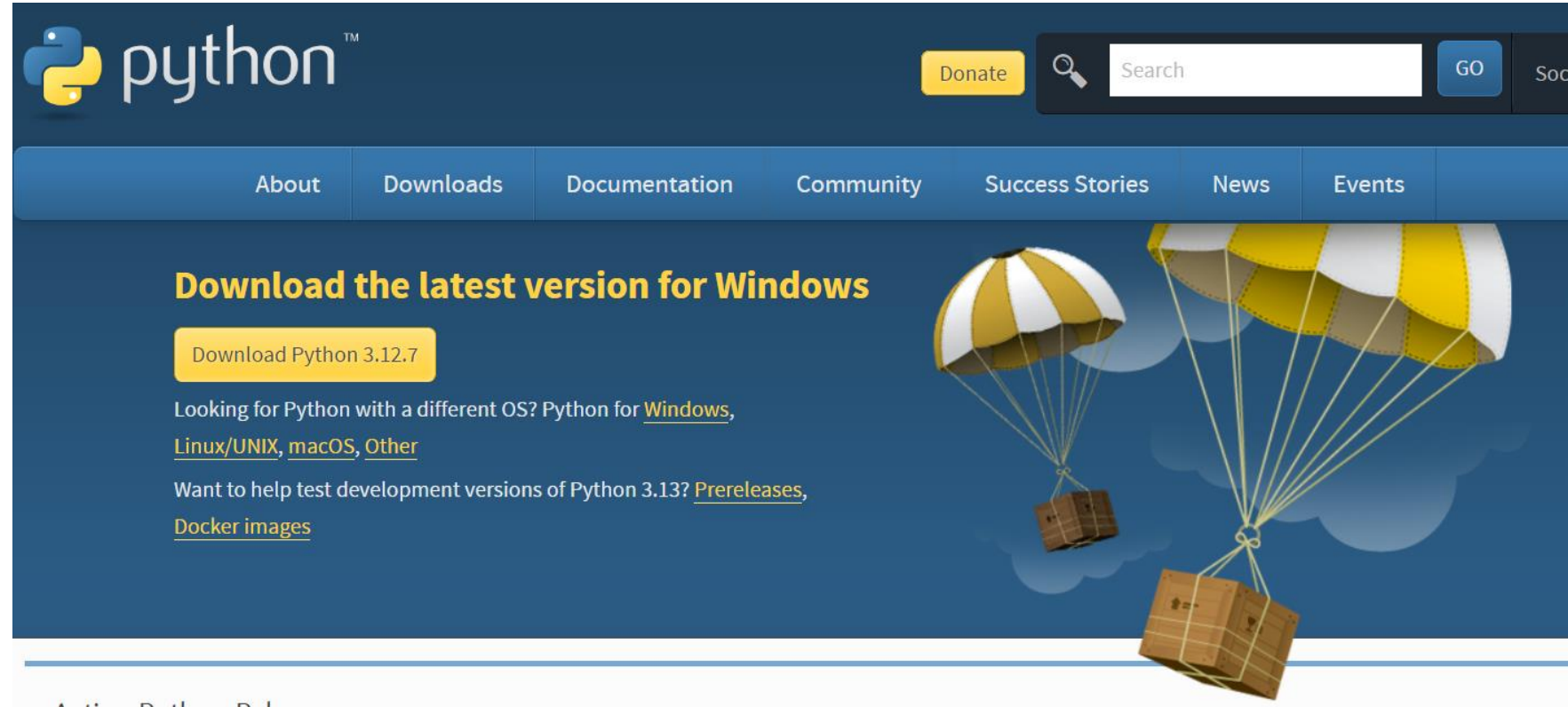
- 1 A Linguagem Python
- 2 Sintaxe Básica e Comentários
- 3 Variáveis
- 4 Tipos de Dados
- 5 Conversões de Dados
- 6 Operadores



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 3.12 é a última versão estável



# 1

## A Linguagem Python

- ☐ Linguagem de alto nível
  - ☐ Componente sintática clara, de fácil leitura e interpretação
  - ☐ Oferece maior abstração dos sistemas computacionais, através de funções, bibliotecas, etc.
- ☐ Implementa paradigmas de:
  - ☐ Programação procedimental, estruturada: módulos, funções, estruturas de dados
  - ☐ Programação orientada a objetos
- ☐ Linguagem multiplataforma (Windows, MacOS, 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

## A Linguagem Python

- ❑ Princípios orientadores da linguagem

### The Zen of Python

```
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

### *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

### Flat is better than nested

"Linear é melhor do que aninhado"

Evite criar estruturas dentro de estruturas que estão dentro de outra estrutura (`dicts` são estruturas por natureza). 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

***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, as vezes inofensivo, as 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
```



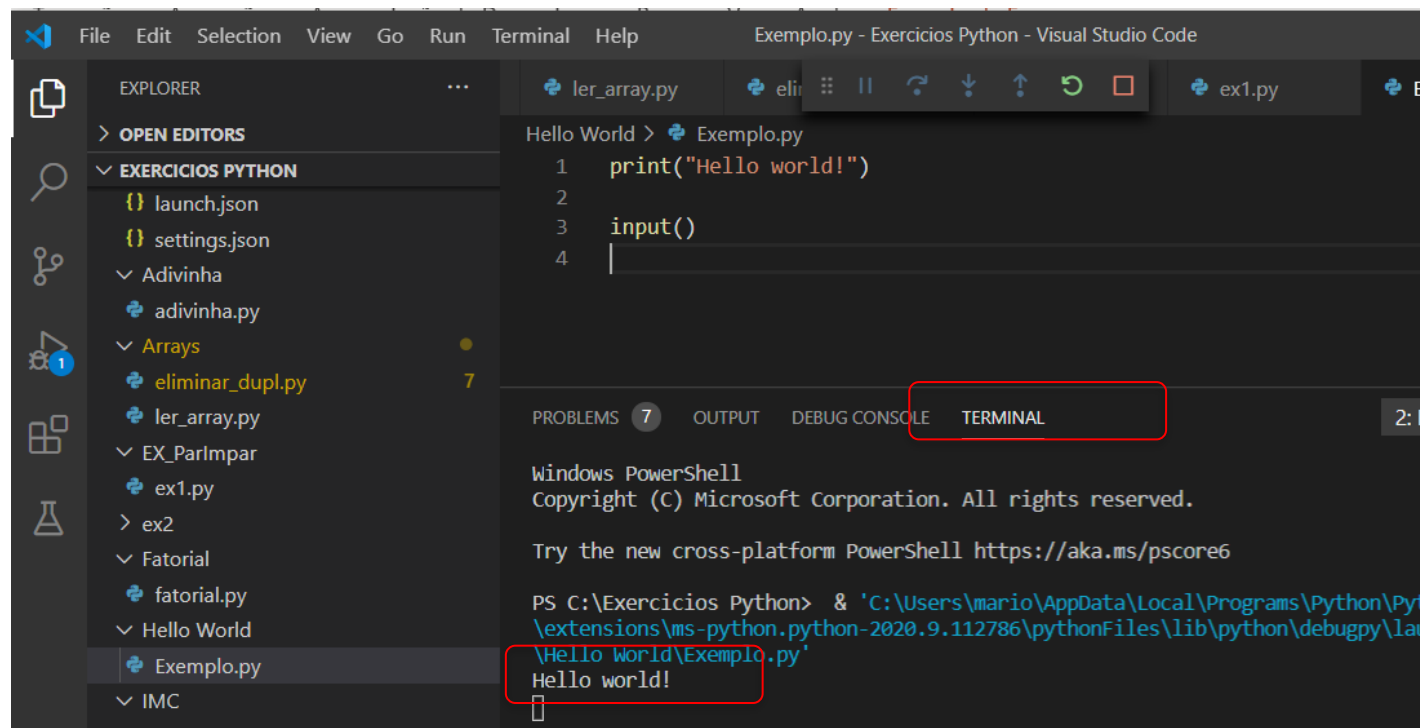
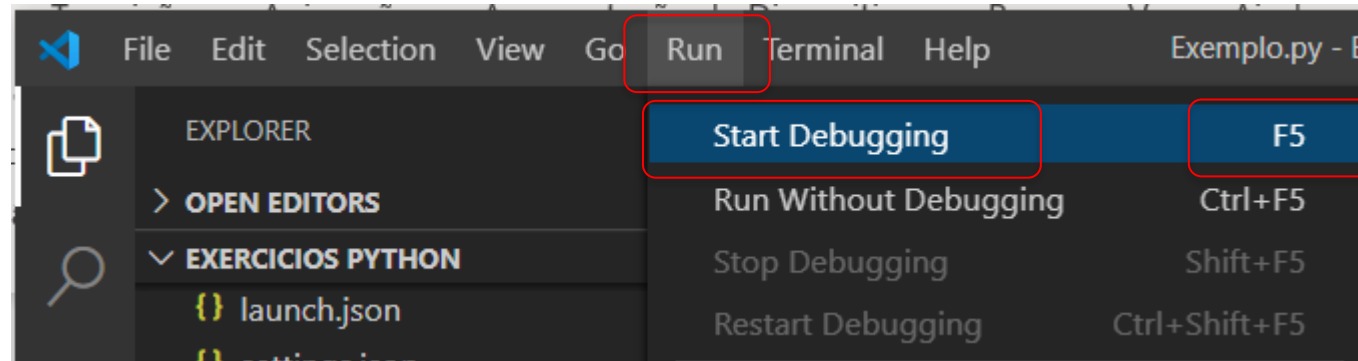
## 2

### Sintaxe básica e comentários

- ❑ Comentários: iniciam-se com `#`
- ❑ Comentar diversas linhas (definir área de comentário) `"""`

```
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
```

## Sintaxe básica e comentários



### 3

## Variáveis

### ☐ VARIÁVEL

Consiste numa estrutura de dados (um objeto) capaz de reter e representar um valor ou uma expressão

☐ Variável pode conter um valor

```
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  
_
```

☐ Variável pode conter uma *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 DEFINIÇÃO 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
```



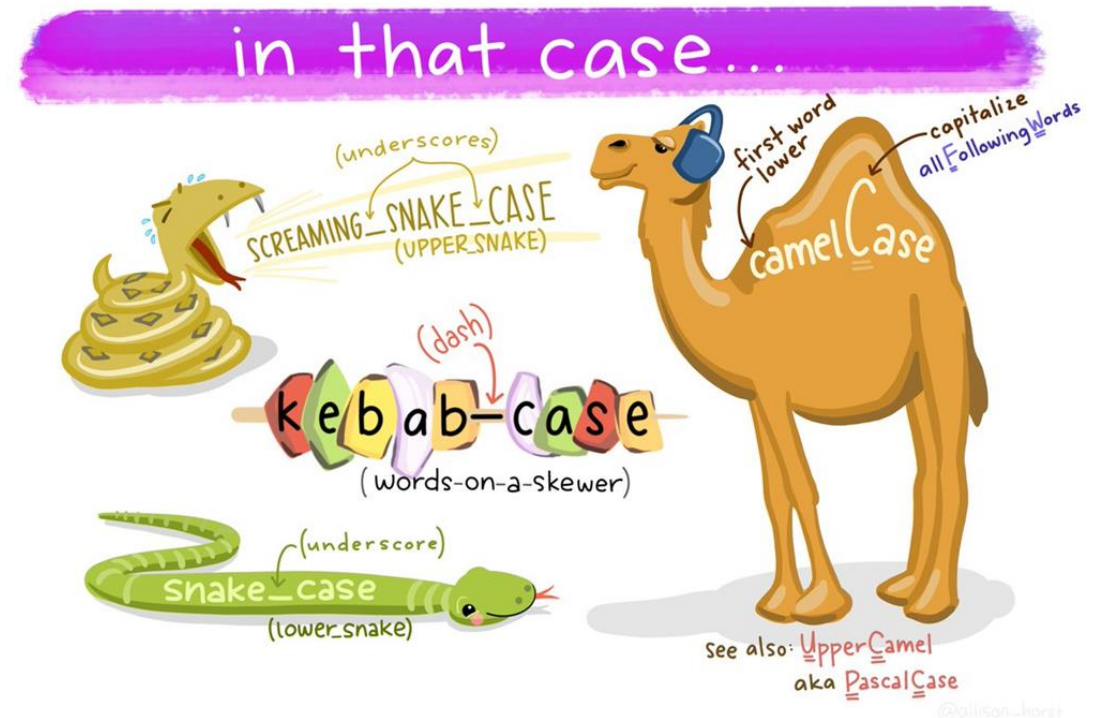
3

## Variáveis

### REGRAS E CONVENÇÕES DE NOMENCLATURA PARA DEFINIÇÃO 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

#### ❑ Convenção camelCase

```
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()
```

#### ❑ Convenção Snake\_case

```
# 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()
```

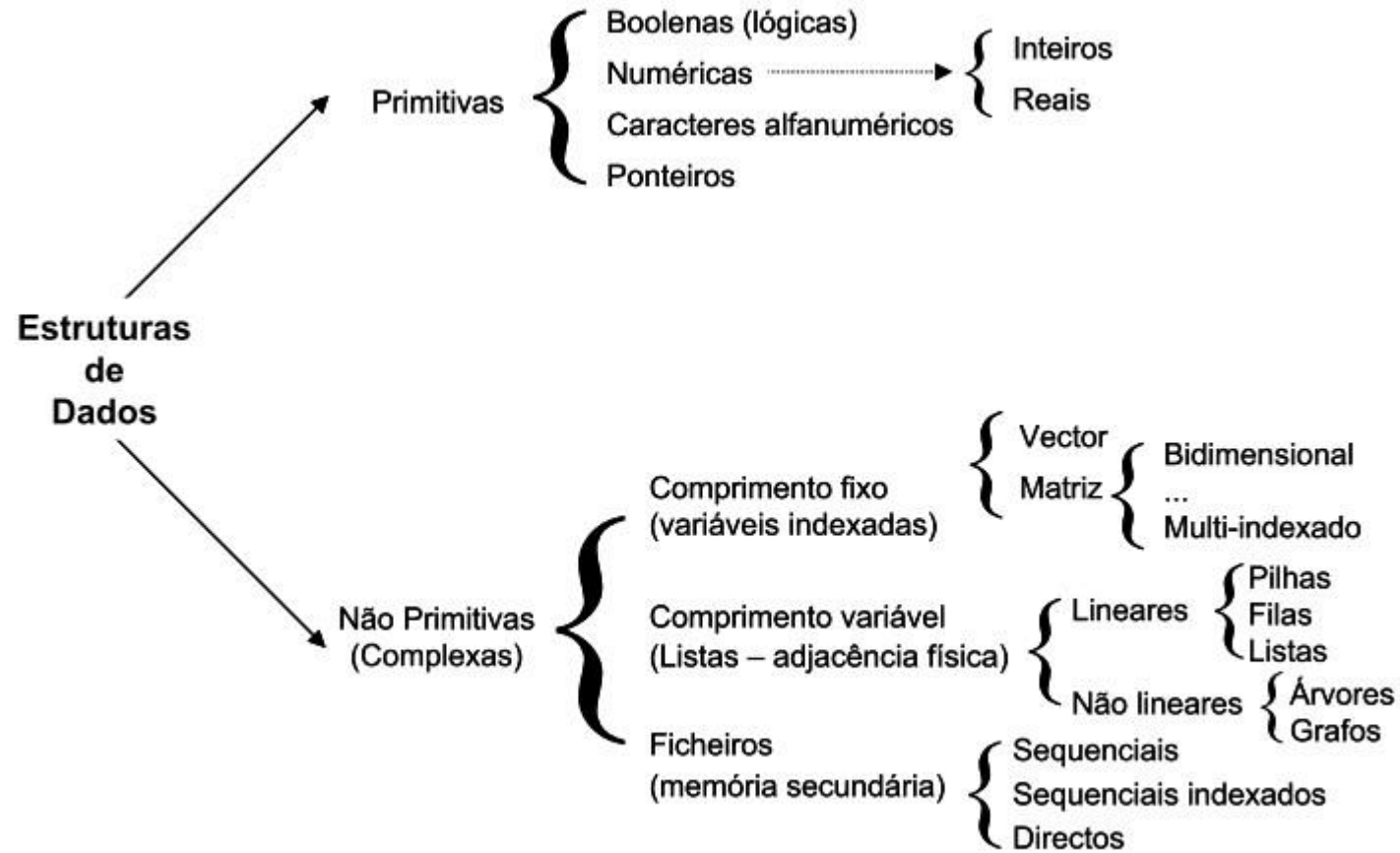
# 3

## Variáveis

### ☐ BOAS PRÁTICAS

- ☐ Usar nomes legíveis *como* *userName*, *totalPagar*, *numProdutos*,...
- ☐ Não usar abreviações ou nomes curtos como a, b, c
- ☐ Nomes descritivos e concisos. Exemplos de nomes inválidos são dados e valor. Esses nomes são genéricos, não dizem nada
- ☐ Ter em consideração os termos usados (regras usadas) pela equipa de desenvolvimento
- ☐ Não misturar idiomas (português com inglês)

## 4 Tipos de dados



4

## Tipos de dados

- ❑ As variáveis podem armazenar diferentes tipos de dados, como dados numéricos, strings, booleanos, sequencias ou coleções.

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



4

## Tipos de dados



```
1
2 # inteiros
3 numero1 = -5
4 numero2 = 0
5 numero3 = 100000
6
7 # float
8 numero1 = -5.0
9 numero2 = 2.8
10 numero3 = 3.14159
11
12 #string
13 msg1 = "Hello World"
14 msg2 = "2.8"
15 msg3 = "54"
16
17 # booleano
18 fimJogo = True
19 fimJogo = False
20
21 # Arrays, listas
22 numeros = [1,48,37,27,18]
23 cidades = ["Porto", "Maia", "Vila do Conde", "Póvoa de Varzim"]
```



```
1 # Dict, objeto dicionário de dados
2 person = {
3     nome: "Carlos",
4     apelido: "Fonseca",
5     idade: 33
6 }
7
8 # tuple
9 capitais = ("Lisboa", "Londres", "Madrid" )
```

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

Conversores	Descrição
<b>Int</b>	Converte para 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, cso contrário devolve erro)
<b>float</b>	Converte para um número flutuante (real). A partir de um literal inteiro, um literal flutuante ou um literal de string (desde que a string represente um flutuante ou um inteiro)
<b>str</b>	Converte para string (texto). A partir de uma ampla variedade de tipos de dados, incluindo strings, literais inteiros e literais flutuantes

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

```
C:\WINDOWS\System32\cmd. x + v
2.82
2
2.82
2.82
Press any key to continue . . . |
```

```
1
2 numero = "28a"
3 print(int(numero))
4
5
```

```
1
2
3 numero = "28a"
4 print(int(numero))

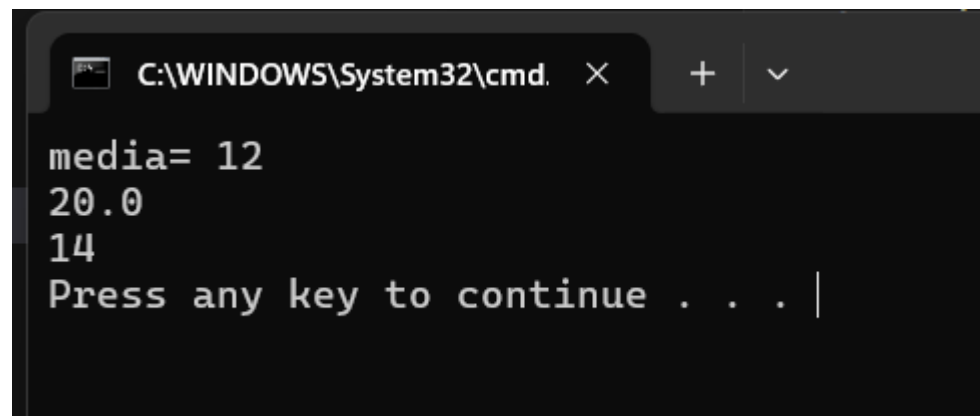
Exception has occurred: ValueError x
invalid literal for int() with base 10: '28a'

File "C:\Users\mario\OneDrive\AED\2023-24\4 -
Exercicios\Ficha 01\temp.py", line 4, in <module>
    print(int(numero))
    ~~~~~^
ValueError: invalid literal for int() with base 10: '28a'
```

5

## Conversões de Dados

```
1  # converte inteiro para string
2  media = 12
3  msg = "media= " + str(media)
4  print(msg)
5
6  # converte inteiro para float
7  numero = 20
8  print(float(numero))
9
10
11 # converte float para inteiro
12 numero = 14.59
13 print(int(numero))
14
```



A screenshot of a Windows Command Prompt window. The title bar shows the path 'C:\WINDOWS\System32\cmd.' and standard window controls. The command prompt displays the output of the Python code: 'media= 12', '20.0', and '14'. It ends with the prompt 'Press any key to continue . . . |'.

```
C:\WINDOWS\System32\cmd.
media= 12
20.0
14
Press any key to continue . . . |
```

## 5

### Conversões de Dados

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

```
1  # converte inteiro para string
2  media = 12
3  msg = "media= " + str(media)
4  print(type(msg))
5
6  # converte inteiro para float
7  numero = 20
8  print(type(float(numero)))
9
10
11 # converte float para inteiro
12 numero = 14.59
13 print(type(int(numero)))
```

```
C:\WINDOWS\System32\cmd.  ×  +  v
<class 'str'>
<class 'float'>
<class 'int'>
Press any key to continue . . . |
```

6

## Operadores

Categoria	Operadores
Aritméticos	+ (soma)
	- (subtração)
	* (multiplicação)
	/ (divisão)
	% (resto da divisão)
	** (exponenciação)
	pow (exponenciação)
	// (divisão truncada)
	abs (valor absoluto)

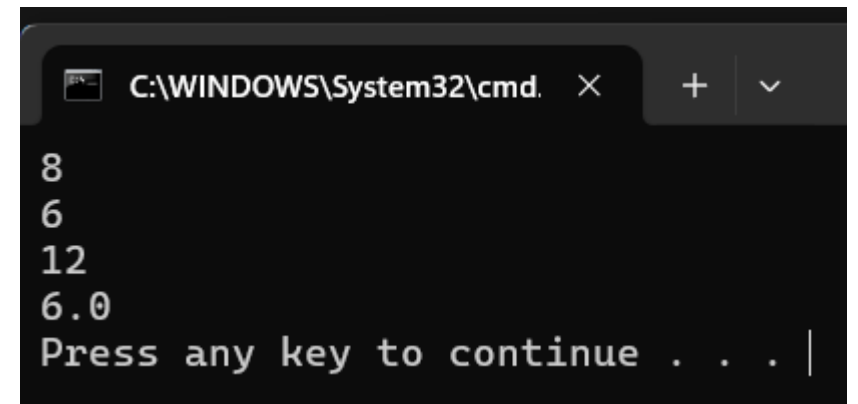
```
numero = 20
numero = numero + 10    # soma
numero = numero / 2     # subtração
numero = numero % 2     # resto da divisão
numero **2              # exponenciação: numero ao quadrado
pow(numero, 2)          # exponenciação: base e expoente

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

6

## Operadores

```
1  # Operações aritméticas usando uma sintaxe tradicional
2  numero=5
3
4  numero = numero + 3
5  print(numero)
6
7  numero = numero -2
8  print(numero)
9
10 numero = numero * 2
11 print(numero)
12
13 numero = numero / 2
14 print(numero)
15
```



C:\WINDOWS\System32\cmd. X + v

8  
6  
12  
6.0  
Press any key to continue . . . |



## 6

## Operadores

- ❑ Algumas **formas abreviadas** de sintaxe de operações aritméticas (muito usadas):

Operator
=
+=
-=
*=
/=
%=
//=
**=

```
1
2  # Operações aritméticas usando uma sintaxe mais abreviada
3  # (mais usual)
4
5
6  numero=5
7
8  numero+=3
9  print(numero)
10
11 numero-=2
12 print(numero)
13
14 numero*=2
15 print(numero)
16
17 numero/= 2
18 print(numero)
```

```
C:\WINDOWS\System32\cmd.  X  +  v
8
6
12
6.0
Press any key to continue . . .
```

## 6 Operadores

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

```
1 numero1 = 10
2 numero2 = 15
3
4 if (numero1 == numero2):
5     print("Os números são iguais")
6
7 if (numero1 is numero2):
8     print("os números são iguais")
9
10 if (numero1 < numero2):
11     print(numero2, "é maior")
12
13 if (numero1 != numero2):
14     print("Os números são diferentes ")
```

C:\WINDOWS\System32\cmd. x + v

15 é maior  
Os números são diferentes  
Press any key to continue . . .

## 6 Entrada e Saída de Dados

### ❑ `input(text)`

*text* – string que representa a mensagem apresentada antes da entrada de dados

```
1
2 # má prática, a introdução de dados não esclarece o que espera do utilizador
3 total = input()
4
5 total = input("Indique o valor total a pagar:")
6
7 total = int(input("Indique o valor total a pagar:"))
8
```

```
C:\WINDOWS\System32\cmd.  X + v
80
Indique o valor total a pagar:110
Indique o valor total a pagar:120
Press any key to continue . . .
```

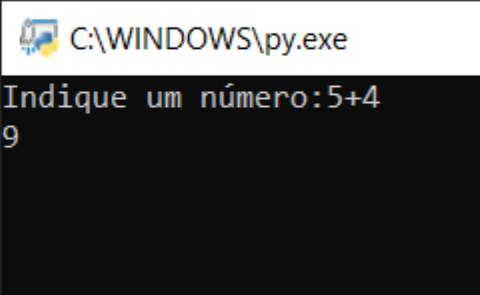
```
1
2 nome = input("Indique o seu nome:")
3
4
```

```
C:\WINDOWS\System32\cmd.  X + v
Indique o seu nome:|
```

## 6 Entrada e Saída de Dados

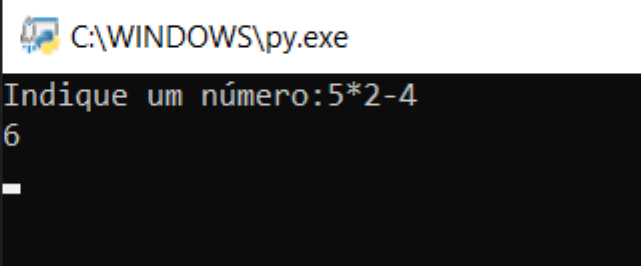
- ❑ `input(text)`  
método ***eval*** (*expression*) – avalia uma expressão

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



C:\WINDOWS\py.exe  
Indique um número:5+4  
9

```
2
3  numero = eval(input("Indique um número:"))
4
5  print(numero)
6
7
8
9
10
```



C:\WINDOWS\py.exe  
Indique um número:5\*2-4  
6

## 6 Entrada e Saída de Dados

❏ `print(texto, variaveis)`

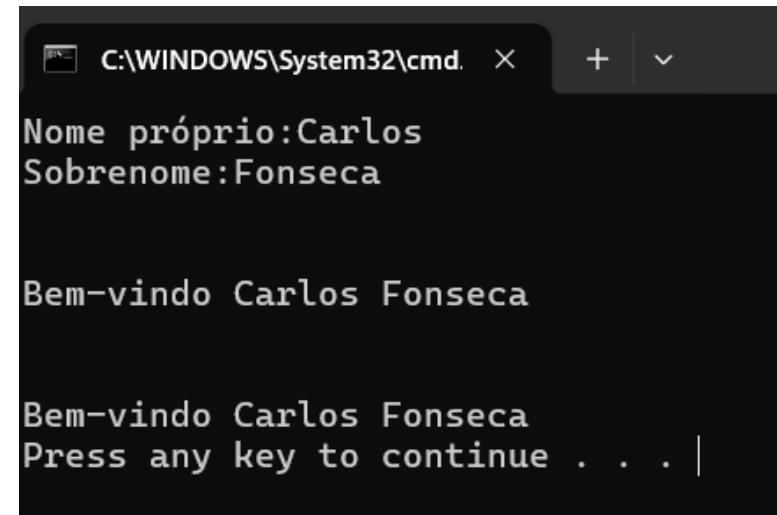
```
1  
2 print("Hello world")  
3 print()  
4  
5 total = int(input("Indique o valor total a pagar:"))  
6 print("O valor com IVA é", total*1.23)  
7  
8 print()  
9  
10 nomeUtilizador = input("nome:")  
11 print("Bem-vindo,", nomeUtilizador)
```

```
EXPLODER  
C:\WINDOWS\System32\cmd. x + v  
Hello world  
  
Indique o valor total a pagar:100  
O valor com IVA é 123.0  
  
nome:Carlos  
Bem-vindo, Carlos  
Press any key to continue . . . |
```

## 6 Entrada e Saída de Dados

❏ `print(texto, variaveis)`

```
1  
2  
3 firstName = input("Nome próprio:")  
4 lastName = input("Sobrenome:")  
5  
6 print("\n")  
7 print("Bem-vindo" + " " + firstName + " " + lastName)  
8 print("\n")  
9 print("Bem-vindo", firstName, lastName)  
10  
11
```



```
C:\WINDOWS\System32\cmd.  X  +  v  
  
Nome próprio:Carlos  
Sobrenome:Fonseca  
  
Bem-vindo Carlos Fonseca  
  
Bem-vindo Carlos Fonseca  
Press any key to continue . . . |
```

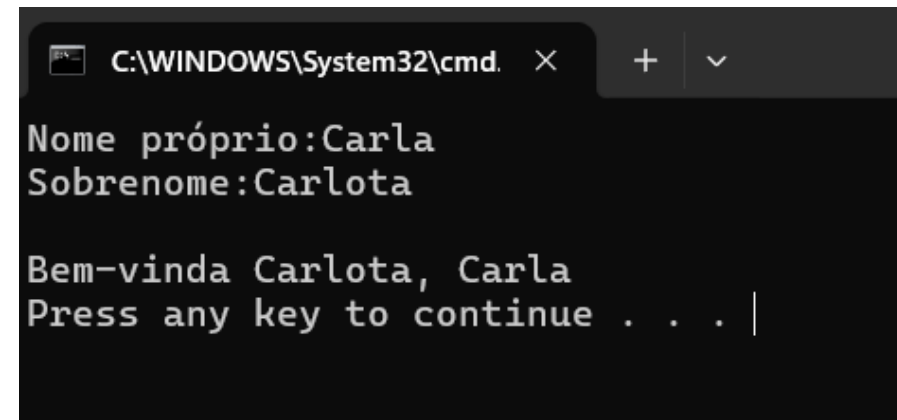


## 6 Entrada e Saída de Dados

- ❑ print()- formatar o *output*:
  - ❑ Usar o operador de string %

```
1
2 firstName = input("Nome próprio:")
3 lastName = input("Sobrenome:")
4
5 print()
6 print("Bem-vinda %s, %s" % (lastName, firstName))
7
8
```

string



```
C:\WINDOWS\System32\cmd.  X  +  v

Nome próprio:Carla
Sobrenome:Carlota

Bem-vinda Carlota, Carla
Press any key to continue . . . |
```

## 6 Entrada e Saída de Dados

- ❑ print()- formatar o *output*:
  - ❑ Usar o operador de string %

float com 2 casas decimais

float com 3 dígitos, sem parte decimal

```
1
2 peso = 63.5
3 altura = 178
4 imc = 20.23
5
6 print("O seu peso é: %.2f, a sua altura é: %3.0f cm" %(peso,altura ))
7
8
```

```
C:\WINDOWS\System32\cmd. x + v
O seu peso é: 63.50, a sua altura é: 178 cm
Press any key to continue . . . |
```

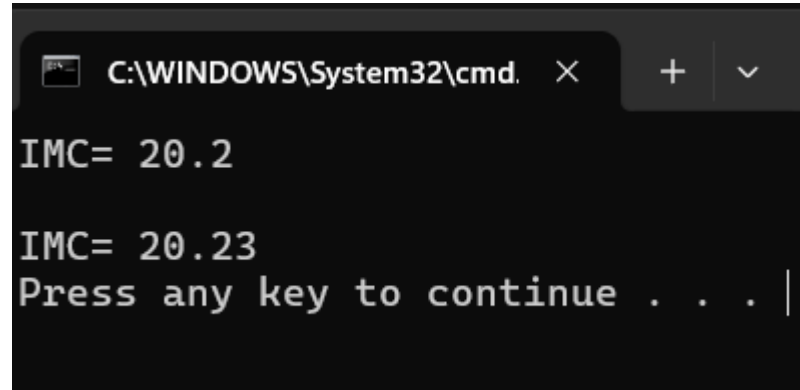
```
1 peso = 63.5
2 altura = 178
3 imc = 20.23
4
5 print("IMC= %.2f" %(imc))
6 print()
7 print("IMC= %.1f" %(imc))
8 print()
9 print("IMC= %5.2f" %(imc))
10 print()
11 print("IMC= %6.2f" %(imc))
12
```

```
C:\WINDOWS\System32\cmd. x + v
IMC= 20.23
IMC= 20.2
IMC= 20.23
IMC= 20.23
Press any key to continue . . . |
```

## 6 Entrada e Saída de Dados

- ❑ print()- formatar o *output*:
  - ❑ Usar o método **format**: **f-strings**

```
1 peso = 63.5
2 altura = 178
3 imc = 20.23
4
5 print("IMC= {:.1f}" .format(imc))
6 print()
7 print("IMC= {:.2f}" .format(imc))
8
```



```
C:\WINDOWS\System32\cmd. × + v
IMC= 20.2
IMC= 20.23
Press any key to continue . . . |
```

## 6 Entrada e Saída de Dados

- ❑ print()- formatar o *output*:
  - ❑ Usar o método **format**: **f-strings**

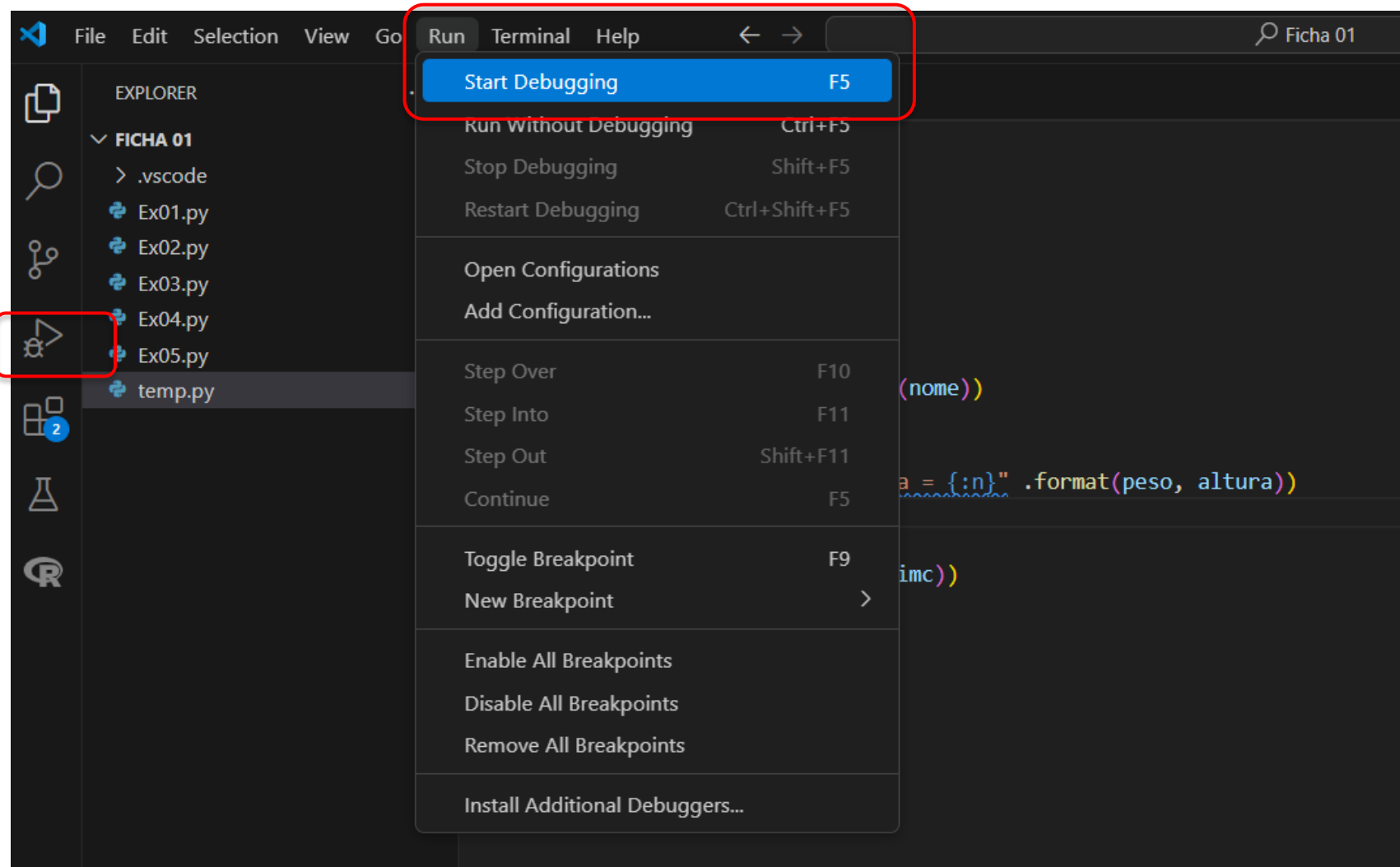
string format      number format

```
1 nome = "Maria Mariazinha"
2 peso = 63.5
3 altura = 178
4 imc = 20.23
5
6 print("Nome: {:s}" .format(nome))
7 print()
8
9 print("Peso= {:.1f}, altura = {:n}" .format(peso, altura))
10 print()
11
12 print("IMC= {:g}" .format(imc))
13
```

general format

```
C:\WINDOWS\System32\cmd. x + v
Nome: Maria Mariazinha
Peso= 63.5, altura = 178
IMC= 20.23
Press any key to continue . . . |
```

## ❑ Executar um programa python



❑ Ficheiro de configuração de pasta com *python files*:

❑ *integratedTerminal*: corre o programa no VSCode, área abaixo do código

❑ *externalTerminal*: corre o programa numa nova janela de execução

