



# Introdução ao Processamento de Dados Turma (2025.1)



## Biblioteca *Math* e classe *String*

Tassio Sirqueira (IME/UERJ)

tassio.sirqueira@ime.uerj.br

# Módulos de Python

- Um **módulo** (*module*) em Python é basicamente uma biblioteca de funções.
- Veremos mais tarde no curso como você pode criar seus próprios módulos.
- Basicamente, um módulo permite que você organize código (programa) escrito em Python.
- Em um módulo pode-se definir **funções**, **classes** e **variáveis**, externas ao núcleo do Python.

# Módulos de Python

- Existem diversos módulos disponíveis, com milhares de funções que não fazem parte do núcleo do Python.
- Diversos módulos já são instalados junto com o Python:

<https://docs.python.org/pt-br/3.12/py-modindex.html>

- Outros módulos muito interessantes devem ser instalados manualmente.
- Exemplos: NumPy; Matplotlib; Scipy; scikit-learn; scikit-image; opencv-python

# Módulos de Python

## NumPy e SciPy

- Computação científica: funções matemáticas complexas; rotinas de álgebra linear; interpolação; transformadas de Fourier, etc.

<https://numpy.org/>

<https://www.scipy.org/>

## Matplotlib

- Visualização: biblioteca para a criação de visualizações estáticas, animadas e interativas.

<https://matplotlib.org/>

# Módulos de Python

## **scikit-learn**

- Aprendizagem de máquina: ferramentas de classificação, regressão, clusterização, pré-processamento de dados, etc.

<https://scikit-learn.org/stable/>

## **scikit-image e opencv-python**

- Diversas rotinas para o processamento de imagens digitais e visão computacional.

<https://scikit-image.org/>

<https://opencv.org/>

# Módulos de Python

- A função ***help*** apresenta a documentação de módulos, funções, classes, palavras reservadas, etc.
- Para saber quais os módulos já estão instalados, use o comando:

```
help("modules")
```

- Para saber o conteúdo de um módulo específico, use o comando:

```
help("nome_do_modulo")
```

- Por exemplo, para mostrar o conteúdo do modulo ***math***:

```
help("math")
```

# Módulos de Python

- Para poder usar as funções de um módulo, é necessário **importá-lo**, usando o comando ***import***.
- Exemplo (importa o módulo ***math***):  

```
import math
```
- A partir daí, você pode usar as funções do módulo, usando como prefixo o nome do módulo.
- Exemplo (cosseno de um ângulo):  

```
math.cos(angulo_em_radianos)
```
- Exemplo (número  $\pi$ ):

```
math.pi
```

# Módulos de Python

- Você também pode dar um apelido ao módulo, para usar no seu programa.
- Exemplo (importa o módulo ***math*** com o nome de ***matematica***):  

```
import math as matematica
```
- A partir daí, você pode usar as funções do módulo, usando como prefixo o apelido do módulo.
- Exemplo (cosseno de um ângulo):  

```
matematica.cos(angulo_em_radianos)
```
- Exemplo (número  $\pi$ ):  

```
matematica.pi
```



# Módulos de Python

- Você também pode importar o módulo, de forma que as funções do módulo podem ser usadas sem o prefixo.
- Exemplo (importa o módulo *math*):  

```
from math import *
```
- A partir daí, você pode usar as funções do módulo, sem ter que colocar um prefixo.
- Exemplo (cosseno de um ângulo):  

```
cos(angulo_em_radianos)
```
- Exemplo (número  $\pi$ ):  

```
pi
```

# Módulo *math*

- O módulo ***math*** contém várias funções matemáticas úteis para resolver problemas matemáticos mais rapidamente.
- Exemplo: funções trigonométricas
  - $\cos(x)$ : calcula o cosseno de  $x$  ( $x$  expresso em radianos)
  - $\sin(x)$ : calcula seno de  $x$  ( $x$  expresso em radianos)
  - $\tan(x)$ : calcula a tangente de  $x$  ( $x$  expresso em radianos)
  - $\arccos(x)$ : calcula o arco cosseno de  $x$  (retorna um valor em radianos)
  - $\arcsin(x)$ : calcula o arco seno de  $x$  (retorna um valor em radianos)
  - $\arctan(x)$ : calcula o arco tangente de  $x$  (retorna um valor em radianos)

# Módulo *math*

Outras funções importantes:

- `log(x,y)`: calcula o logaritmo de  $x$  na base  $y$
- `factorial(x)`: calcula o fatorial de  $x$
- `sqrt(x)`: calcula a raiz quadrada de  $x$
- `abs(x)`: retorna o valor absoluto de um inteiro  $x$
- `fabs(x)`: retorna o valor absoluto de um real  $x$
- `ceil(x)`: retorna o menor número inteiro maior do que o real  $x$
- `floor(x)`: retorna o maior número inteiro menor do que o real  $x$

## Módulo *math*

Exemplo: programa que lê um número ( $n$ ) e calcula o valor da seguinte expressão

$$\sum_{i=1}^n \sqrt{i}$$

A expressão equivale a:

$$\sqrt{1} + \sqrt{2} + \sqrt{3} + \sqrt{4} + \dots + \sqrt{n}$$

## Módulo *math*

Exemplo: programa que lê um número ( $n$ ) e calcula o valor da seguinte expressão

$$\sum_{i=1}^n \sqrt{i}$$

```
import math
n = int(input('Entre com um número: '))
somatorio = 0
for i in range (1,n+1):
    somatorio = somatorio + math.sqrt(i)
print('Valor do somatório:', somatorio)
```

# Cadeias de caracteres (*strings*)

- As *strings* são formadas por caracteres da tabela **ASCII** (ou **UNICODE**).

# Codificação de textos

## Tabela ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

# Cadeias de caracteres (*strings*)

- As *strings* são formadas por caracteres da tabela **ASCII** (ou **UNICODE**).
- Para transformar um valor de qualquer outro tipo de dados em *string* basta usar o comando:

`str(valor)`

- Exemplos:

`str(True)` *retorna* 'True'

`str(3.14)` *retorna* '3.14'



# Cadeias de caracteres (*strings*)

- Todas as classes/tipos que possuem a ideia de ordenação (como a classe ***string***) podem ser acessadas por partes.
- Cada elemento da ***string*** possui um número (índice) em sequência.
- Exemplo: nome = 'PYTHON'

<b>P</b>	<b>Y</b>	<b>T</b>	<b>H</b>	<b>O</b>	<b>N</b>
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

# Cadeias de caracteres (*strings*)

Exemplo: nome = 'PYTHON'

P	Y	T	H	O	N
0	1	2	3	4	5

- A letra 'P' na variável (***string***) **nome** equivale a: nome[0]
- A letra 'N': na variável **nome** equivale a: nome[5]
- A sequência de caracteres 'YTH' na variável **nome** equivale a: nome[1:4]
- Os 5 primeiros caracteres 'PYTHO' na variável **nome** equivalem a:  
nome[0:5] ou nome[:5]
- Para pegar os caracteres alternados ('PTO'):  
nome[0:5:2] ou nome[:5:2]

# Cadeias de caracteres (*strings*)

- Você pode concatenar ***strings*** usando o sinal de +
- Exemplos:

```
nome = 'Gil' + 'son'  
numero_hexadecimal = '1' + 'A'
```

- Você pode descobrir o tamanho de uma ***string*** usando o comando ***len(...)***.
- Exemplos:

```
len('turma 3 de IPD')  
len(numero_hexadecimal)
```

# Cadeias de caracteres (*strings*)

- A função ***ord(...)*** recebe uma ***string*** formada por um **único caractere** e retorna seu número na tabela ASCII.
- Exemplos:

`ord('a')` *retorna* 97

`ord('!')` *retorna* 33

`ord(' ')` *retorna* 32

# Cadeias de caracteres (*strings*)

- A função ***chr(...)*** recebe um número inteiro e retorna o caractere correspondente na tabela ASCII.
- Exemplos:

*chr(97) retorna 'a'*

*chr(33) retorna '!'*

*chr(32) retorna ' '*

# Métodos da Classe *string*

- Na realidade uma cadeia de caracteres (constante ou variável) é uma **instância** (objeto) da **classe *string***.
- Estes são conceitos da metodologia de **programação orientada a objetos**.
- Não vamos entrar em detalhes sobre programação orientada a objetos neste curso, basta saber que uma classe de objetos contém **métodos**.
- Métodos são muito parecidos com funções, mas se aplicam a um objeto da classe, e tem a seguinte sintaxe:

`objeto.nome_do_metodo(...)`

# Métodos da Classe *string*

Exemplos de métodos da classe ***string***:

- `count(...)` – conta o número de ocorrências de uma *sub-string*.
- `lower( )` – transforma todos os caracteres em minúsculos.
- `upper( )` – transforma todos os caracteres em maiúsculos.
- `isalpha( )` – indica se todos os caracteres são letras.
- `isdigit( )` – indica se todos os caracteres são dígitos (números).
- `replace(...)` – substitui uma *sub-string* por outra.
- `split(...)` – quebra uma *string* quando encontra uma determinada *sub-string*.

<http://www.python-ds.com/python-3-string-methods>

# Métodos da Classe *string*

***count(...)*** – conta o número de ocorrências de uma *sub-string*.

- Sintaxe:

```
string_a.count(string_b,inicio,fim)
```

- Conta quantas vezes *string\_b* aparece dentro de *string\_a*.
- *string\_a* e *string\_b* podem ser constantes ou variáveis.
- *inicio* e *fim* são opcionais, e indicam os limites (índices) da busca dentro de *string\_a*.



# Métodos da Classe *string*

***count(...)*** – conta o número de ocorrências de uma *sub-string*.

- Sintaxe:

```
string_a.count(string_b,inicio,fim)
```

- Exemplos:

```
a = 'abacaxi'
```

```
a.count('a', 0, 3) retorna 3
```

```
a.count('ac') retorna 1
```

```
a.count('ix') retorna 0
```

# Métodos da Classe *string*

***lower()*** – transforma todos os caracteres em minúsculos.

***upper()*** – transforma todos os caracteres em maiúsculos.

- Sintaxe:

`nome_string.lower()`

`nome_string.upper()`

- Retornam versões em maiúsculas ou minúsculas da *nome\_string*.
- *nome\_string* pode ser uma constante ou variável.
- Os métodos não têm parâmetros.

# Métodos da Classe *string*

***lower()*** – transforma todos os caracteres em minúsculos.

***upper()*** – transforma todos os caracteres em maiúsculos.

- Sintaxe:

nome\_string.lower()

nome\_string.upper()

- Exemplos:

a = 'Rio de Janeiro'

a.lower() *retorna* 'rio de janeiro'

a.upper() *retorna* 'RIO DE JANEIRO'

# Métodos da Classe *string*

*isalpha( )* – indica se todos os caracteres são letras.

*isdigit( )* – indica se todos os caracteres são dígitos (números).

- Sintaxe:

`nome_string.isalpha()`

`nome_string.isdigit()`

- *nome\_string* pode ser uma constante ou variável.
- Os métodos não têm parâmetros.

# Métodos da Classe *string*

*isalpha( )* – indica se todos os caracteres são letras.

*isdigit( )* – indica se todos os caracteres são dígitos (números).

- Sintaxe:

nome\_string.isalpha()

nome\_string.isdigit()

- Exemplos:

a = '0123456789'

a.isalpha() *retorna* False

a.isdigit() *retorna* True

# Métodos da Classe *string*

*isalpha( )* – indica se todos os caracteres são letras.

*isdigit( )* – indica se todos os caracteres são dígitos (números).

- Sintaxe:

nome\_string.isalpha()

nome\_string.isdigit()

- Exemplos:

a = 'Pindamonhangaba'

a.isalpha() *retorna* True

a.isdigit() *retorna* False

# Métodos da Classe *string*

- Exercício: programa que lê uma *string* e imprime o total de letras, o total de números e a quantidade de outros caracteres.

```
cadeia = input('Entre com uma string: ')
contA = contN = contO = 0
for i in range(0, len(cadeia)):
    if cadeia[i].isalpha():
        contA += 1
    elif cadeia[i].isdigit():
        contN += 1
    else:
        contO += 1
print('Letras: ', contA, 'Números: ', contN, 'Outros: ', contO)
```

# Métodos da Classe *string*

***replace(...)*** – substitui uma *sub-string* por outra.

- Sintaxe:

`nome_string.replace(sub_a, sub_b, qte)`

- *nome\_string*, *sub\_a* e *sub\_b* podem ser constantes ou variáveis.
- Substitui as ocorrências de *sub\_a* em *nome\_string* por *sub\_b*.
- O parâmetro *qte* é opcional: indica quantas vezes a substituição deve ser feita.



# Métodos da Classe *string*

***replace(...)*** – substitui uma *sub-string* por outra.

- Sintaxe:

nome\_string.replace(sub\_a, sub\_b, qte)

- Exemplo:

a = 'Brasil'

a.replace('s', 'z') *retorna* 'Brazil'

b = 'Pindamonhangaba'

b.replace('a', '\_', 2) *retorna* 'Pind\_monh\_ngaba'

# Métodos da Classe *string*

***split(...)*** – quebra uma *string* quando encontra uma determinada *sub-string*.

- Sintaxe:

`nome_string.split(sub_a, qte)`

- *nome\_string* e *sub\_a* podem ser constantes ou variáveis.
- Retorna uma lista de *strings*.
- O parâmetro *qte* é opcional: indica quantas vezes a quebra deve ser feita.

# Métodos da Classe *string*

***split(...)*** – quebra uma *string* quando encontra uma determinada *sub-string*.

- Sintaxe:

```
nome_string.split(sub_a, qte)
```

- Exemplo:

```
cadeia = 'Pindamonhangaba'
```

```
cadeia.split('a', 2) retorna ['Pind', 'monh', 'ngaba']
```

```
texto = 'Rio de Janeiro'
```

```
texto.split(' ') retorna ['Rio', 'de', 'Janeiro']
```

# Métodos da Classe *string*

- Exercício: programa que lê uma *string* (frase) e diz se ela contém uma determinada palavra.

```
palavra = 'vovó'
qte = 0
texto = input('Entre com uma frase: ')
lista_de_palavras = texto.split(' ')
for p in lista_de_palavras:
    if p.upper() == palavra.upper():
        qte = qte + 1
if qte > 0:
    print('Palavra', palavra, 'encontrada', qte, 'vezes.')
else:
    print('Palavra', palavra, 'não encontrada.')
```

## Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

```
>>> print('O número é {}'.format(13))
```

```
>>> O número é 13.
```

```
>>> numero = 13
```

```
>>> frase = 'O número é: {}.'
```

```
>>> print(frase.format(numero))
```

```
>>> O número é 13.
```

## Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

```
>>> print('Primeira palavra: {}; segunda palavra: {}'.format('um', 'dois'))
```

```
>>> 'Primeira palavra: um; segunda palavra: dois.'
```

```
>>> print('Primeira palavra: \'{}\'; segunda palavra: \'{}\'.format('um', 'dois'))
```

```
>>> Primeira palavra: 'um'; segunda palavra: 'dois'.
```

## Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Inserindo uma nova linha (\n):

```
>>> print('Primeira palavra: \'{}\'' \nSegunda palavra: \'{}\'''.format('um', 'dois'))
```

```
>>> Primeira palavra: 'um'
```

```
>>> Segunda palavra: 'dois'
```

## Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Um número dentro das chaves pode indicar a posição de relativa de cada valor passado como parâmetro do método *format*:

```
>>> print('Primeira palavra: {0}. Segunda palavra: {1}'.format('um', 'dois'))
```

```
>>> Primeira palavra: um. Segunda palavra: dois.
```

```
>>> print('Primeira palavra: {1}. Segunda palavra: {0}'.format('um', 'dois'))
```

```
>>> Primeira palavra: dois. Segunda palavra: um.
```



# Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Preenchimento (*padding*) e alinhamento:
- String resultante tem 10 caracteres. Alinhamento à direita.

```
>>> '{:>10}'.format('teste')
```

```
>>> '    teste'
```

- String resultante tem 10 caracteres. Alinhamento centralizado.

```
>>> '{:^10}'.format('teste')
```

```
>>> ' teste '
```

# Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Preenchimento (*padding*) e alinhamento:
- String resultante tem 10 caracteres. Alinhamento à esquerda.

```
>>> '{:<10}'.format('teste')
```

```
>>> 'teste  '
```

```
>>> '{:10}'.format('teste')
```

```
>>> 'teste  '
```

# Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Preenchimento (padding) e alinhamento:
- Preenchimento padrão é com espaços em branco, mas outro caractere pode ser usado.

```
>>> '{:_<10}'.format('teste')
```

```
>>> 'teste_____'
```

```
>>> '{:_^10}'.format('teste')
```

```
>>> '__teste____'
```

# Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Truncando strings longas.

```
>>> '{:.6}'.format('Rio de Janeiro')
```

```
>>> 'Rio de'
```

- Truncando e preenchendo.

```
>>> '{:*^10.3}'.format('Rio de Janeiro')
```

```
>>> '***Rio***'
```

# Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Formatando números inteiros.

```
>>> '{:d}'.format(42)
```

```
>>> '42'
```

```
>>> '{:10d}'.format(42)
```

```
>>> '      42'
```

```
>>> '{:010d}'.format(42)
```

```
>>> '0000000042'
```

- O número a ser formatado tem que ser inteiro, senão dá erro!

## Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Formatando números reais (*float*).

```
>>> '{:f}'.format(3.141592653589793)
```

```
>>> '3.141593'
```

```
>>> '{:.2f}'.format(3.141592653589793)
```

```
>>> '3.14'
```

```
>>> '{:5.2f}'.format(3.141592653589793)
```

```
>>> ' 3.14'
```

```
>>> '{:05.2f}'.format(3.141592653589793)
```

```
>>> '03.14'
```

# Métodos da Classe *string*

***format(...)*** – Insere um valor (transformado em *string*) dentro da *string* a ser formatada, na posição definida por { }.

- Números com sinal positivo (o negativo sempre aparece).

```
>>> '{:+d}'.format(42)
```

```
>>> '+42'
```

```
>>> '{:+d}'.format(-42)
```

```
>>> '-42'
```

```
>>> '{:d}'.format(-42)
```

```
>>> '-42'
```



# Introdução ao Processamento de Dados Turma (2025.1)



## Biblioteca *Math* e classe *String*

Tassio Sirqueira (IME/UERJ)

tassio.sirqueira@ime.uerj.br