Python Pro - notas de curso

PYTHON BIRDS

Contents

[1-Programação Procedural 2](#_Toc44598874)

[1.1-Introdução 2](#_Toc44598875)

[1.1.1-Motivação 2](#_Toc44598876)

[1.1.2-Instalação Windows 2](#_Toc44598877)

[1.1.3-Instalação Ubuntu 2](#_Toc44598878)

[1.1.4-Instalação Mac 2](#_Toc44598879)

[1.1.5-Console Interativo 2](#_Toc44598880)

[1.1.6-Pycharm IDE 2](#_Toc44598881)

[1.2-Tipos Básicos 3](#_Toc44598882)

[1.3-Containers e Iteração 4](#_Toc44598883)

[1.3.1-Container String, Dir e Help + pliques/aspas simples, duplas, triplas \n lower upper swapcase strip 4](#_Toc44598884)

[1.3.2-Container Lista e Range + append, pop, extend, split, join, clear, reverse 6](#_Toc44598885)

[1.3.3-Container "Tupla" e Id 9](#_Toc44598886)

[1.3.4-Acesso, Tamanho e Fatiamento (slice) + índices 11](#_Toc44598887)

[1.3.5-While (laço) 13](#_Toc44598888)

[1.3.6-For 14](#_Toc44598889)

[1.3.7-Dicionários/Mapas (ou hashtable) 17](#_Toc44598890)

[1.3.8-Iteração em Dicionário 18](#_Toc44598891)

[1.4-Modularização 20](#_Toc44598892)

[1.4.1-Função e PEP 8 20](#_Toc44598893)

[1.4.2-Parâmetros de Função 21](#_Toc44598894)

[1.4.3-Parâmetros Variáveis 22](#_Toc44598895)

[1.4.4-Módulo (e execução) 25](#_Toc44598896)

[1.4.5-Debug 25](#_Toc44598897)

[1.4.6-Import e \_\_name\_\_ 25](#_Toc44598898)

[1.4.7-Pacote 26](#_Toc44598899)

[1.4.8-Docstring e Comentário 27](#_Toc44598900)

[1.4.9-Contagem de Caracteres com Lista 28](#_Toc44598901)

[1.4.10-Contagem de Caracteres com Dicionário 30](#_Toc44598902)

[1.4.11-Retrospectiva: Paradigma Procedural 31](#_Toc44598903)

[2-Orientação a Objetos 32](#_Toc44598904)

[2.1-Classe e Composição 32](#_Toc44598905)

[2.2-Herança 32](#_Toc44598906)

# 1-Programação Procedural

## 1.1-Introdução

### 1.1.1-Motivação

==> Git no Windows

### 1.1.2-Instalação Windows

### 1.1.3-Instalação Ubuntu

### 1.1.4-Instalação Mac

### 1.1.5-Console Interativo

No Windows, chame por py -3 ; em sistemas Unix, chame por **python3**

O console também é conhecido por REPL (Read Eval Print Loop)

">>>" indica que o console está esperando a expressão, Enter envia a mesma, quando o Python avaliar e, não havendo erro, apresenta o resultado

“...” significa...

<https://docs.python-requests.org/en/master> - nele usa a tela do console como documentação!

### 1.1.6-Pycharm IDE

IDEs procuram conter todas ferramentas necessárias pra desenvolver seu projeto

## 1.2-Tipos Básicos

## 1.3-Containers e Iteração

Containers contém uma seqüência de objetos

### 1.3.1-Container String, Dir e Help + pliques/aspas simples, duplas, triplas \n lower upper swapcase strip

String ou ‘cadeia de caracteres’ pode ser definida com aspas simples.

**>>> 'Marcelo'**

'Marcelo'

**>>> type('')**

<class 'str'>

**>>> "Mellissa"**

'Mellissa'

**>>> type("")**

<class 'str'>

**>>> 'Arthur''**

SyntaxError: EOL while scanning string literal

**>>> 'Marcelo\'' 🡸 legibilidade ruim**

"Marcelo'"

**>>> "Mell's"**

"Mell's"

**>>> 'Python \nPro' 🡸 legibilidade comprometida**

'Python \nPro'

**>>> print(\_)**

Python

Pro

**>>> '''Python**

**Pro'''**

'Python\nPro'

**>>> print(\_)**

Python

Pro

**>>> '''Marcelo '"''' 🡸 uma aspas simples, uma aspa dupla, três aspas simples fechando a string**

'Marcelo \'"'

**>>> type('''e''')**

<class 'str'>

**>>> 'Python '+'Pro'**

'Python Pro'

**>>> 'o'\*4**

'oooo'

**>>> 'Marcelo'\*4**

'MarceloMarceloMarceloMarcelo' **🡸 útil para validar campo que impeça string maior que 60 caracteres...**

**>>> 'Ipsum Lorem '\*5 🡸 ...ou para criar textos ‘Ipsum Lorem’**

'Ipsum Lorem Ipsum Lorem Ipsum Lorem Ipsum Lorem Ipsum Lorem '

**>>> dir('') 🡸 passa uma lista de operações e atributos que este objeto** [string] **possui**

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', **'lower'**, 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', **'upper'**, 'zfill']

**>>> help('Marcelo'.lower) 🡸 SEM O ABRE e FECHA PARÊNTESES / ajuda a entender os componentes de objeto e suas mensagens de ajuda**

Help on built-in function lower:

lower() method of builtins.str instance

Return a copy of the string converted to lowercase.

**>>> 'Marcelo'.lower()**

'marcelo'

**>>> 'Mell'.upper()**

'MELL'

**>>> 'MArcelo'.swapcase()**

'maRCELO'

**>>> ' Mellissa Miranda '.strip()**

'Mellissa Miranda'

**>>> '\_\_\_Arthur\_Miranda\_\_\_\_\_'.strip('\_')**

'Arthur\_Miranda'

### 1.3.2-Container Lista e Range + append, pop, extend, split, join, clear, reverse

Lista seria um ‘vetor dinâmico’ em outras linguagens, e significa que pode acrescentar ou remover elementos dela.

**>>> [1,2,3] 🡸 com COLCHETES e separados por VÍRGULAS**

[1, 2, 3]

**>>> type([])**

<class 'list'>

**>>> lista = list(range(10)) 🡸 cria lista de 0 a 9**

**>>> lista**

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista = list(range(1, 10)) 🡸 começando de 1 e indo até 10**

**>>> lista**

[1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista = list(range(1, 10, 2)) 🡸 lembra de P.A.** (progressão aritmética)**?**

**>>> lista**

[1, 3, 5, 7, 9]

**>>> lista = list(range(10, 0, -2)) 🡸 P.A. com razão negativa** (decrescente)

**>>> lista**

[10, 8, 6, 4, 2]

**>>> dir(lista)**

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', **'sort'**]

**>>> lista.sort() 🡸 coloca na ordem crescente**

**>>> lista**

[2, 4, 6, 8, 10]

**>>> lista.append(12) 🡸 acrescenta elemento ao final da lista**

**>>> lista**

[2, 4, 6, 8, 10, 12]

**>>> lista.pop() 🡸 remove ultimo elemento indicando qual foi este**

12

**>>> lista**

[2, 4, 6, 8, 10]

**>>> splited\_text**

['Return', 'a', 'copy', 'of', 'the', 'string', 'with', 'leading', 'and', 'trailing', 'whitespace', 'removed.', 2020, 22]

**>>> splited\_text.sort()**

Traceback (most recent call last):

File "<pyshell#27>", line 1, in <module>

splited\_text.sort()

TypeError: '<' not supported between instances of 'int' and 'str'

**>>> lista.extend([12,14])**

**>>> lista**

[2, 4, 6, 8, 10, 12, 14]

**>>> lista + [16,18]**

[2, 4, 6, 8, 10, 12, 14, 16, 18]

**>>> [1,3]\*3**

[1, 3, 1, 3, 1, 3] **🡸 repetindo 3 vezes**

**>>> 'Python Pro'.split() 🡸 ‘espaço’ é o separador *default***

['Python', 'Pro']

**>>> 'Python-Pro'.split('-') 🡸 para o separador como ‘-‘** (hífen) *\* vide split no dir(string) na seção 1.3.1*

['Python', 'Pro']

**>>> lista = \_**

['Python', 'Pro']

**>>> '#'.join(lista) 🡸 junta com trecho ‘#’** (*hashtag*) *\* vide join no dir(string) na seção 1.3.1*

'Python#Pro'

**>>> "\_#\_".join(lista)**

'Python\_#\_Pro'

**>>> [1, 1.0, 'Marcelo', []] 🡸 a lista pode conter quaisquer objetos juntos**

[1, 1.0, 'Marcelo', []]

**>>> lista03 = list(range(6)) + ["boi"]**

**>>> lista03**

[0, 1, 2, 3, 4, 5, 'boi']

**>>> lista04 = list(range(6)) + ['Boi', 'Galinha']**

**>>> lista04**

[0, 1, 2, 3, 4, 5, 'Boi', 'Galinha']

**>>> lista.clear()**

**>>> lista**

[]

**>>> lista=list(range(10))**

**>>> lista.insert(0, -1) 🡸 insere -1 antes do valor 0 na lista**

**>>> lista**

[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**Abaixo formas que não funcionam..!**

**>>> lista.insert(0, range(-5,-2,-1))**

**>>> lista**

[**range(-5, -2, -1)**, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista.insert(0, (range(-5,-2,-1)))**

**>>> lista**

[**range(-5, -2, -1)**, range(-5, -2, -1), -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista=list(range(10))**

**>>> lista.insert(0, -1) 🡸 insere -1 antes do valor 0 na lista**

**>>> lista**

[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista.insert(0, [-5, -4, -3, -2])**

**>>> lista**

[**[-5, -4, -3, -2]**, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista.insert(0, (-5, -4, -3, -2))**

**>>> lista**

[**(-5, -4, -3, -2)**, [-5, -4, -3, -2], -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista.insert(0, -5, -4, -3, -2)**

Traceback (most recent call last):

File "<pyshell#121>", line 1, in <module>

lista.insert(0, -5, -4, -3, -2)

TypeError: insert expected 2 arguments, got 5

**…uma forma de resolver isto seria**

**>>> lista=list(range(10))**

**>>> lista.insert(0, -1)**

**>>> lista**

[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista.reverse() 🡸 …inverter a lista!**

**>>> lista**

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1]

…e acrescentar seus números na ordem decrescente!

**>>> lista.append(range(-5,-2,-1)) 🡸 porém com append não funciona!**

**>>> lista**

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, **range(-5, -2, -1)**]

**>>> lista.remove(range(-5, -2, -1)) 🡸 removendo primeira ocorrência de range(-5-2,-1)**

**>>> lista**

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1]

**>>> lista.extend([-2,-3,-4,-5]) 🡸 …mas sim com extend**

**>>> lista**

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5]

**>>> lista.extend(range(-6,-10,-1)) 🡸 …inclusive com range**

**>>> lista**

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9]

### 1.3.3-Container "Tupla" e Id

**>>> tpl = (1, 2) 🡸 elementos separados por vírgula**

**>>> type (tpl)**

<class 'tuple'>

Do Fórum: em tuplas quem constrói as tuplas é a vírgula ", " - o operador **construtor** de tuplas é a vírgula.

Os parênteses servem para agrupar elementos para ajudar n a legibilidade e agrupar os elementos da tupla.

**>>> t1 = 1,2,3,4,5**

**>>> t2 = (1,2,3,4,5)**

**>>> t3 = 1,**

**>>> type(t1)**

<class 'tuple'>

**>>> type(t2)**

<class 'tuple'>

**>>> type(t3)**

<class 'tuple'>

**Referência:** [**Built-in Types — Python 3.8.2 documentation 1**](https://docs.python.org/3/library/stdtypes.html#tuple) - <https://docs.python.org/3/library/stdtypes.html#tuple>

Diferente da lista, a tupla é IMUTÁVEL

**>>> dir(tpl) 🡸 não existem métodos para listas, que modificam o conteúdo da lista**

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'count', 'index']

**>>> tuple(range(6)) 🡸 FUNÇÃO tuple**

(0, 1, 2, 3, 4, 5)

Com relação à criação de tuplas, existe também precedência de operadores (como de operadores matemáticos)

**>>> (6) 🡸 pegadinha de criação de tuplas... não seria tupla mas um INTEIRO**

6 🡸 se trata de inteiro

**>>> type((6))**

<class 'int'> 🡸 confirma que (6) se trata de inteiro

**>>> (6,) 🡸 para caracterizar uma tupla de um elemento é preciso colocar uma vírgula**

(6,) 🡸 **tupla de apenas um elemento**

**>>> type((6,))**

<class 'tuple'>

**>>> [6,] 🡸 pode ser usado vírgula na criação de uma lista**

[6] **\* na linguagem precursora do Python (ABC) a tuple era chamada de compound**

**>>> registro=('Marcelo', 50)**

**>>> nome, idade = registro 🡸 "desempacotamento" da lista colocando os elementos na ordem**

**>>> nome**

'Marcelo'

**>>> idade**

50

**>>> nome, idade, id = registro 🡸 com variáveis adicionais ao número de elementos da lista incorre em erro**

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

nome, idade, id = registro

ValueError: not enough values to unpack (expected 3, got 2)

**>>> nome, idade = (1, 2, 3) 🡸 ...e com número de elementos da lista maior que o de variáveis também dá erro**

Traceback (most recent call last):

File "<pyshell#10>", line 1, in <module>

nome, idade = (1, 2, 3)

ValueError: too many values to unpack (expected 2)

**>>> registro\_2=('Luciano', 40)**

**>>> registro + registro\_2**

('Marcelo', 50, 'Luciano', 40)

**>>> registro2 = registro\*2**

**>>> registro2**

('Marcelo', 50, 'Marcelo', 50)

**>>> nome, idade = registro2**

Traceback (most recent call last):

File "<pyshell#151>", line 1, in <module>

nome, idade = registro2

ValueError: too many values to unpack (expected 2)

**>>> nome1, idade1, nome2, idade2 = registro2**

**>>> nome1**

'Marcelo'

**>>> idade1**

50

**>>> nome2**

'Marcelo'

**>>> idade2**

50

**>>> id(registro) 🡸 para verificar a identidade de um objeto**

67028872

**>>> id(registro\_2)**

67203912

**>>> id(registro+registro\_2)**

67743208

Artigo interessante: **Tuplas mutantes em Python** <http://pythonclub.com.br/tuplas-mutantes-em-python.html> >> [https://docs.python.org/3/reference/datamodel.html#objects-values-and-types](https://docs.python.org/3/reference/datamodel.html" \l "objects-values-and-types)

### 1.3.4-Acesso, Tamanho e Fatiamento (slice) + índices

\* acesso aos elementos da string, lista e tupla; e fatiamento para gerar subsequencias

**>>> nome='Marcelo'**

**>>> nome[0]**

'M'

**>>> nome[1]**

'a'

**>>> len(nome)**

7

**>>> nome[len(nome)-1] 🡸 lembre-se que os índices dos elementos começam em 0**

'o'

**>>> nome[len(nome)]**

Traceback (most recent call last):

File "<pyshell#27>", line 1, in <module>

nome[len(nome)]

IndexError: string index out of range

**>>> nome[-1]**

'o' 🡸 **índices negativos trazem na ordem inversa (do fim pro início)**

**>>> nome[-2]**

'l'

Fatiamento / *Slicing*

**>>> nome[0:3]**

'Mar' 🡸 **intervalo aberto de 0 (primeiro) até o de índice 2 (terceiro)**

**>>> nome[1:3]**

'ar'

**>>> nome[-3:len(nome)]**

'elo' 🡸 **apresenta três últimos caracteres da string**

**>>> nome[-3:] 🡸 ...ou também desta forma**

'elo'

**>>> nome[:3] 🡸 ...ou 3 de seu início**

'Mar'

**>>> nome[:4:2]**

'Mr' 🡸 **quatro primeiros caracteres de dois em dois**

**>>> nome[::-1]**

'olecraM' 🡸 **ao contrário (de -1 em -1)**

**>>> nome[::2]**

'Mreo' 🡸 **caracteres de dois em dois**

...também serve para listas!

**>>> lista=list(range(10))**

**>>> lista**

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>> lista[0]**

0

**>>> lista[-1]**

9

**>>> lista[:3] 🡸 slicing**

[0, 1, 2]

**>>> lista[::-1] 🡸 slicing com passo**

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

**>>> len(lista)**

10

**>>> lista[::2]**

[0, 2, 4, 6, 8]

**>>> lista[0:10:2]**

[0, 2, 4, 6, 8]

**Comentário do Renzo no Fórum em 12 Set 2019 :**

Performance é algo que vc só se preocupa no final do processo. Os passos são:

1. Resolver o problema
2. Deixar a solução mais legível
3. Homologar a solução para ver se atende o cliente

No caso de não atender o cliente por causa de performance:

1. Medir a aplicação para encontrar o gargalo, ou seja, qual componente possui maior parcela de responsabilidade pela baixa performance
2. Atuar para melhorar a performance do gargalo.
3. Voltar para o passo 3.

Isso elucida a célebre frase do Knut: **“Otimização precoce é raiz de todos males”.**

### 1.3.5-While (laço)

**>>> nome='Marcelo'**

**>>> i = 0**

**>>> while i<len(nome):**

**print(nome[i])**

M

M

M

M

M

M

M

M

M

...

**>>> while i<len(nome):**

**print(nome[i])**

**i +=1**

M

a

r

c

e

l

o

**Do Fórum 25 Mai 2020 :** possibilidade de uso do 'else em um laço while:

**while <condição>: codigo else: código**

O código no else será executado APENAS se não ocorrer nenhuma chamada a break no código do while … se todas as iterações ocorrerem sem a chamada a break, ao sair do while o código do else será executado.

**...resposta do Renzo: (...)** Mas a forma que acho bem bacana de memorizar o else é que se a condição usada no while for “verdadeira”, o else é executado. Serve também para o for. E justamente costuma ser o break que faz com que vc saia do laço com a condição usada nele ainda sendo verdadeira.

### 1.3.6-For

**>>> nome='Mel'**

**>>> i = 0**

**>>> while i<len(nome):**

**print(nome[i])**

**i +=1**

M

e

l

…muitas vezes se esquece que o índice começa em zero, ou esquecemos de incrementar a variável, além da legibilidade comprometida por conta da variável ( i ) servir apenas como cursor!

**>>> for v in nome:**

**print(v)**

M

e

l

…podemos imprimir o índice – note porém que a legibilidade também fica comprometida!

**>>> for i in range(len(nome)):**

**print(i, nome[i])**

0 M

1 e

2 l

**>>> for i, v in enumerate(nome):**

**print(i, nome[i])**

0 M

1 e

2 l

**>>> for i, v in enumerate(nome):**

**print(i, v)**

0 M

1 e

2 l

**>>> cidade='Teresina'**

**>>> for c, v in enumerate(cidade, start=1)**

**print(c, v)**

File "<input>", line 1

for c, v in enumerate(cidade, start=1) 🡸 **faltou o dois-pontos ( : )**

^

SyntaxError: invalid syntax

**>>> cidade=’Rio’**

**>>> for c, v in enumerate(cidade, start=4):**

**print(c, v)**

**4 R**

**5 i**

**6 o**

...um pouco mais sobre **enumerate**.

**>>> enumerate('Marcelo')**

<enumerate object at 0x038A3608>

**>>> print(enumerate('Marcelo'))**

<enumerate object at 0x038A42A8>

**>>> a = enumerate('Marcelo')**

**>>> print(a)**

<enumerate object at 0x038A4048>

**>>> help(enumerate)**

Help on class enumerate in module builtins:

class enumerate(object)

| enumerate(iterable, start=0)

|

| Return an enumerate object.

|

| iterable

| an object supporting iteration

|

| The enumerate object yields pairs containing a count (from start, which

| defaults to zero) and a value yielded by the iterable argument.

|

| enumerate is useful for obtaining an indexed list:

| (0, seq[0]), (1, seq[1]), (2, seq[2]), ...

|

| Methods defined here:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_next\_\_(self, /)

| Implement next(self).

|

| \_\_reduce\_\_(...)

| Return state information for pickling.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

**>>> type(enumerate('Marcelo'))**

<class 'enumerate'>

…do <https://docs.python.org/3/library/functions.html?highlight=enumerate#enumerate>

**enumerate**(iterable, start=0)[¶](https://docs.python.org/3/library/functions.html?highlight=enumerate#enumerate)

Return an enumerate object. iterable must be a sequence, an [iterator](https://docs.python.org/3/glossary.html" \l "term-iterator), or some other object which supports iteration. The [\_\_next\_\_()](https://docs.python.org/3/library/stdtypes.html#iterator.__next__) method of the iterator returned by [enumerate()](https://docs.python.org/3/library/functions.html?highlight=enumerate#enumerate) returns a tuple containing a count (from start which defaults to 0) and the values obtained from iterating over iterable.

>>>

**>>>** seasons = ['Spring', 'Summer', 'Fall', 'Winter']

**>>>** list(enumerate(seasons))

[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]

**>>>** list(enumerate(seasons, start=1))

[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]

Equivalent to:

**def** enumerate(sequence, start=0):

n = start

**for** elem **in** sequence:

**yield** n, elem

n += 1

### 1.3.7-Dicionários/Mapas (ou hashtable)

...ou hashtable.Busca conectar chaves a identificadores!

**>>> linguas = {'br':'portugues', 'eua':'ingles'}**

**>>> type(linguas)**

<class 'dict'>

**>>> linguas**

{'br': 'portugues', 'eua': 'ingles'}

**>>> linguas['br']**

'portugues' 🡸 para chave ‘br’

**>>> linguas['eua']**

'ingles' 🡸 para chave ‘eua’

**>>> linguas['es']**

Traceback (most recent call last):

File "<pyshell#93>", line 1, in <module>

linguas['es']

KeyError: 'es'

...pode acessar os elementos com o método GET onde também passa a chave!

**>>> linguas.get('es')** 🡸 note que não recebe erro! Mas sim um null

**>>> print(linguas.get('es'))**

None

**>>> linguas.get('es','nao definida') 🡸 para informar ‘não definida’ quando o container NÃO contém a chave!**

'nao definida'

**>>> linguas.get('br','nao definida')**

'portugues'

**>>> 'br' in linguas**

True

**>>> 'cn' in linguas**

False

**>>> 6 in list(range(10))**

True

**>>> 11 in list(range(10))**

False

**>>> linguas['es'] = 'spanhol'**

**>>> linguas['es']**

'spanhol'

**>>> linguas**

{'br': 'portugues', 'eua': 'ingles', 'es': 'spanhol'}

**>>>>>> linguas['es']='espanhol'**

**>>> linguas**

{'br': 'portugues', 'eua': 'ingles', 'es': 'espanhol'}

### 1.3.8-Iteração em Dicionário

**>>> for chave in linguas:**

**print(chave)**

br

eua

es

**>>> for chave in linguas.keys:**

**print(chave)**

br

eua

es

**>>> for chave in linguas.keys():**

**print(chave)**

br

eua

es

**>>> for valor in linguas.values():**

**print(valor)**

portugues

ingles

espanhol

**>>> for chave, valor in linguas.items():**

**print(chave, valor)**

br portugues

eua ingles

es espanhol

...no dicionário, a partir da versão 3.6 do Python, os itens de um dicionário ficam ordenados segundo sua inserção!

\* em um dicionário não faz sentido falar em ordenação (garantida no Python 3.6)

**>>> linguas.pop('br') 🡸 tem que explicitamente dizer a CHAVE para remover o respectivo item**

'portugues'

**>>> linguas**

{'eua': 'ingles', 'es': 'espanhol'}

**>>> del linguas['eua'] 🡸 não retorna o valor apagado**

**>>> linguas**

{'es': 'espanhol'}

Do Fórum: tem como criar um dicionário com mais de 2 indices?

Sim, pode se utilizar dicionários dentro de dicionários. Por exemplo:

**times = {**

**'cruzeiro': {**

**'serie':'B',**

**'observacao':'maior de minas'**

**},**

**'atletico':{**

**'serie':'A',**

**'observacao':'segundomaior de minas'**

**}**

**}**

**>>> print(times.keys())**

['cruzeiro', 'atlético']

**>>> for time, dados in times.items():**

**... print(time, dados)**

**...**

cruzeiro {'serie': 'B', 'observacao': 'maior de minas'}

atletico {'serie': 'A', 'observacao': 'segundomaior de minas'}

LISTAS - entre colchetes: { 1, 2, 3 ] ou [ 1, 2, ‘Marcelo’, True ]

TUPLAS – entre parênteses: (1, 2, 3)

DICIONÁRIOS – entre chave: {

## 1.4-Modularização

### 1.4.1-Função e PEP 8

Nesse tópico será abordado como construir uma função. Além disso você vai aprender o que é uma PEP (Python Enhancement Purposal) em particular a 8, que é um guia de estilos para escrita de código Python.

**. Naming Conventions** [**https://www.python.org/dev/peps/pep-0008/**](https://www.python.org/dev/peps/pep-0008/)

nomes de variáveis e funções em "snake case"

**>>> def ola\_mundo():**

**return 'Olá, Mundo!'**

**>>> type(ola\_mundo)**

<class 'function'>

**>>> ola\_mundo**

<function ola\_mundo at 0x037FB580>

**>>> ola\_mundo()**

'Olá, Mundo!'

**>>> def ola\_mundo():**

**pass 🡸 não passa nada**

**>>> resultado=ola\_mundo()**

**>>> resultado 🡸 não traz nada!**

**>>> type(resultado)**

<class 'NoneType'> 🡸 **apesar disso, tem um tipo associado!**

\* em outras linguagens 'None' se chama 'Null'

**>>> print(resultado)**

None

### 1.4.2-Parâmetros de Função

Nesse tópico você vai aprender mais detalhes sobre os parâmetros de função: como definir valores padrão, como passar valores pode nome em vez de justaposição. Como item extra, ainda vai aprender o básico sobre f-strings, uma cadeia de caracteres que permite utilizar valores de variáveis em sua construção.

**>>> def ola(nome):**

**return f'Olá, {nome}'**

**>>> ola('Marcelo')**

'Olá, Marcelo'

**>>> def ola(nome, sobrenome):**

**return f'Olá, {nome} {sobrenome}'**

**>>> ola('Mellissa','Miranda') # Justaposição**

'Olá, Mellissa Miranda'

**>>> def ola(nome, sobrenome='Miranda'): 🡸 definido valor *default***

**return f'Olá, {nome} {sobrenome}'**

**>>> ola('Arthur')**

'Olá, Arthur Miranda'

**>>> ola('Marcelo','Espasandim')**

'Olá, Marcelo Espasandim'

**>>> def ola(nome, sobrenome='Miranda', idade=50):**

**return f'Olá, {nome} {sobrenome} {idade}'**

**>>> ola('Marcelo')**

'Olá, Marcelo Miranda 50' 🡸 **com só um parâmetro os outros usam o *default***

**>>> ola('Mellissa', 13)**

'Olá, Mellissa 13 50'

**>>> ola('Mellissa', idade=13) 🡸 não mais por justaposição, mas por nome, e então a ordem deixa de ser relevante**

'Olá, Mellissa Miranda 13'

**>>> ola('Arthur', idade=8, sobrenome='N. M.')**

'Olá, Arthur N. M. 8'

### 1.4.3-Parâmetros Variáveis

Nesse tópico você vai aprender a criar uma função com um número variável de argumentos, tanto por justaposição quanto por nome.

**>>> def soma(\*parcelas):**

**print(parcelas)**

**print(type(parcelas))**

**>>> soma()**

()

<class 'tuple'>

**>>> soma(1)**

(1,)

<class 'tuple'>

**>>> soma(1,2)**

(1, 2)

<class 'tuple'>

**>>> def soma(\*parcelas):**

**aux = 0**

**for valor in parcelas:**

**aux += valor**

**return aux**

**>>> soma()**

0

**>>> soma(2)**

2

**>>> soma(2,4)**

6

**>>> soma(2, 4, 10)**

16

**>>> def f(\*\*kwargs): # 🡸 kwargs : keyword arguments**

**print(kwargs)**

**print(type(kwargs))**

**>>> f()**

{}

<class 'dict'>

**>>> f(1)**

Traceback (most recent call last):

File "<pyshell#115>", line 1, in <module>

f(1)

TypeError: f() takes 0 positional arguments but 1 was given

**>>> f(nome = 'Marcelo')**

{'nome': 'Marcelo'}

<class 'dict'>

**>>> f(nome='Marcelo',sobrenome='Miranda')**

{'nome': 'Marcelo', 'sobrenome': 'Miranda'}

<class 'dict'>

**>>> args=(2,4,10)**

**>>> kwargs={'nome': 'Marcelo', 'sobrenome': 'Miranda'}**

**>>> def f(\*args, \*\*kwargs): # 🡸 primeiro os variáveis por justaposição (args) e depois os nomeados (kwargs)**

**print(args)**

**print(kwargs)**

**>>> f()**

()

{}

**>>> f(1,2, nome='Arthur', sobrenome='Miranda')**

(1, 2)

{'nome': 'Arthur', 'sobrenome': 'Miranda'}

**>>> f(args, kwargs)**

((2, 4, 10), {'nome': 'Marcelo', 'sobrenome': 'Miranda'})

{}

...tenho que fazer o contrário do que fiz acima e utilizar o desempacotamento

**>>> f(\*args)**

(2, 4, 10)

{}

**>>> f(\*\*kwargs)**

()

{'nome': 'Marcelo', 'sobrenome': 'Miranda'}

**>>> f(\*args, \*\*kwargs)**

(2, 4, 10)

{'nome': 'Marcelo', 'sobrenome': 'Miranda'}

**...explicando empacotamento e desempacotamento de parâmetros!**

**DESEMPACOTAMENTO**

# função para listar itens  
def listar\_itens(w, x, y, z):  
 print(x, w, y, z)  
  
lista = [21, 22, 67, 69]  
  
# tentando listar os itens diretamente:  
listar\_itens(lista)

...a execução cai em erro

C:\Users\Celo\PycharmProjects\novo\venv\Scripts\python.exe C:/Users/Celo/PycharmProjects/novo/exercicios/Pkg1.4.3\_empacotamento.py

Traceback (most recent call last):

File "C:/Users/Celo/PycharmProjects/novo/exercicios/Pkg1.4.3\_empacotamento.py", line 8, in <module>

listar\_itens(lista)

TypeError: listar\_itens() missing 3 required positional arguments: 'x', 'y', and 'z'

Process finished with exit code 1

…corrigindo o código, usando o **\*** que indica que o argumento deve ser DESEMPACOTADO

# função para listar itens  
def listar\_itens(w, x, y, z):  
 print(x, w, y, z)  
  
lista = [21, 22, 67, 69]  
  
# tentando listar os itens com o operador adequado:  
listar\_itens(\*lista)

...execução sem erros

22 21 67 69

**EMPACOTAMENTO**

# empacotando itens  
# - ref.: https://www.youtube.com/watch?v=1TbOk\_r3eYc : 5:45 empacotamento de parametros  
def somar(\*args): # preparada para trabalhar com número indefinido de argumentos  
 soma = 0  
 for i in range(0, len(args)):  
 soma += args[i]  
 return soma  
  
# testando  
print(somar(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))  
print(somar(77, 23))

...funcionou ok!

55

100

### 1.4.4-Módulo (e execução)

Botão díreito do mouse sobre NOVO > New / Python File

PyCharm > File / Settings / Keymap / - Pesquisar por:

"increase font size” aumenta tamanho da fonte / mouse right button > Ctrl + / OK / REMOVE

"decrease"… reduz… / mouse right button > Ctrl - / OK / REMOVE

**Ctrl Shift F12** para entrar no modo de edição, apenas com o código a ser editado

**Help / Keymap Reference** com todos atalhos existentes

**def soma(parcela, parcela\_2):**

**return parcela + parcela\_2**

**print(soma(1,2)**

No quadro da parte inferior da tela...

Aba Terminal > dir >> python matematica.py

mouse right button >> run 'matematica'

### 1.4.5-Debug

…alguns enchem o código de print para avaliar seu código!

Inserção/Remoção de Breakpoints

mouse right button >> debug 'matematica'

**Shift F8** step out

**F7** step into

**F8** step over

selecionando trecho do código / mouse right button >> **Add to Watches**

o andamento pode ser verificado na aba 'Console'

### 1.4.6-Import e \_\_name\_\_

Módulo **executar\_matematica**

**def soma (parcela, parcela\_2):**

**return parcela + parcela\_2**

**# print(\_\_name\_\_) # "dunder name"**

**# evitar que o código de teste de um módulo seja executado quando dentro de outros módulos**

**if \_\_name\_\_=="\_\_main\_\_":** 🡸 truque para identificar o módulo

**print(soma(1,2))**

...quando executando diretamente o arquivo matematica.py, \_\_name\_\_ será igual a ‘\_\_main\_\_’

### 1.4.7-Pacote

nos pacotes aparece o \_\_init\_\_.py **🡸 não mexo!**

\matematica\executar\_matematica.py

\matematica\ matematica.py

**import matematica**

**print (matematica.soma(4,5))**

**...não funcionará!**

\matematica\executar\_matematica.py

\matematica\**mat.py** 🡸 altera nome do antigo matematica.py

**import matematica.mat**

**print (matematica.mat.soma(4,5))**

...pode então rodar no PyCharm com **botão direito / Run > executar\_matematica.py**

ou no Terminal com a linha de comando **python –m matematica.executar\_matematica**

...como ficou muito grande **matematica.met.soma(...)** uma forma de encurtar isso é usar como abaixo

**from matematica import mat**

**print (mat.soma(4,5))**

...ou ainda

**from matematica import soma**

**print (soma(4,5))**

...ou também

**from matematica. mat import soma as s**

**print(s(4, 5))**

...criando uma pasta aninhada chamada **base** com o arquivo mat.py

\matematica\executar\_matematica.py

\matematica\ **base\**mat.py

**from matematica.base.mat import soma as s 🡸 refatorado por conta de mover o mat.py para a pasta base**

**print(s(4, 5))**

### 1.4.8-Docstring e Comentário

**# < comentário**

... também pode ser feito com Ctrl / para ativa ou desativar comentários

**DocString – documentação do módulo, disponível no help deste...**

"Módulo que contém operações Matemáticas" (entre aspas – simples ou duplas)

...se usar três aspas duplas e apertar **Enter**, ele completa automaticamente com um *template* para documentação!

**def soma(parcela, parcela\_2):**

**"""Essa função calcula a soma de duas parcelas**

**:param parcela:** number

**:param parcela\_2:** number

**:return:** number

**"""**

**return parcela + parcela\_2**

**# print(\_\_name\_\_) # "dunder name" <- truque para identificar o módulo**

**# evitar que o código de teste de um módulo seja executado**

**# quando dentro de outros módulos**

**if \_\_name\_\_=="\_\_main\_\_":**

**print(soma(1,2))**

na aba Python Console:

**>>> from matematica.base import mat**

**>>> help(mat)** 🡸 traz a DocString do módulo mat

rodando Rerun (**Ctrl F5**) - ícone no canto superior esquerdo do quadro Python Console:

**>>> from matematica.base import mat**

**>>> help(mat.soma)** 🡸 traz a DocString da função soma

...também pode ser trazido o DocString com o “dunder doc” (**\_\_doc\_\_**)

**>>> mat.soma.\_\_doc\_\_**

'Essa função calcula a soma de duas parcelas\n\n :param parcela:\n :param parcela\_2:\n :return:\n '

...o caractere especial \n representa os ‘pulos’ de linha.

### 1.4.9-Contagem de Caracteres com Lista

Nesse tópico será resolvido o clássico problema de entrevistas de emprego de contagem de caracteres. Nessa primeira versão o problema será resolvido com uma lista.

Apenas com a resolução de exercícios e construção de projetos reais você conseguirá se colocar no mercado de trabalho; conhecimento em computação entre pelos dedos e não pelos ouvidos!

**Ctrl D** no PyCharm duplica a linha digitada; e Ctrl Shift ↓ para mover esta para a linha de baixo

Exercício comum de processos seletivos:

**def contar\_caracteres(s):**

**"""Função que conta os caracteres de uma string**

**Ex:**

**>>> contar\_caracteres('Marcelo')**

**a: 1**

**c: 1**

**e: 1**

**l: 1**

**M: 1**

**o: 1**

**r: 1**

**>>> contar\_caracteres('banana')**

**a: 3**

**b: 1**

**n: 2**

**:param s: string a ser contada**

**"""**

**caracteres\_ordenados = sorted(s)**

**caracter\_anterior = caracteres\_ordenados[0]**

**contagem = 1**

**for caracter in caracteres\_ordenados[1:]:**

**if caracter==caracter\_anterior:**

**contagem += 1**

**else:**

**print(f'{caracter\_anterior}: {contagem}')**

Usando os conceitos do **import** mostrados no tópico sobre Pacotes, no Python Console.

**>>> from exercicios.contagem\_de\_caracteres\_lista import contar\_caracteres as contar**

**>>> contar('Mellissa')**

**M: 1**

**a: 1**

**e: 1**

**i: 1**

**l: 2**

**s: 2**

**Do Fórum:**

**josue23 Membro Fev '19**

**>>> sorted('Josué')**

**['J', 'o', 's', 'u', 'é']**

**>>> sorted('Josue')**

**['J', 'e', 'o', 's', 'u']**

Pelo que eu entendi o sorted() usa as posições da tabela ASCII pra ordenar como mostra os dois exemplos acima.

Um tem acento na letra e, o outro não tem acento.

Eu procurei uma maneira de ordenar alfabeticamente mas não encontrei.

Alguém tem uma dica de documentação?

**renzo Membro Fev '19**

Na realidade ele ordena utilizando a tabela Unicode. Para saber o codepoint (código de um caracter na tabela unicode:

**>>> ord(u'e')**

**101**

**>>> ord(u'é')**

**233**

Por acaso essa tabela segue os mesmos números que a codificação ASCII, mas a tabela unicode é muito maior, o objetivo é ter todos caracteres do mundo:

**http://unicode.org/charts/ 1**

**renzo Membro Fev '19**

Eu procurei uma maneira de ordenar alfabeticamente mas não encontrei.

Alguém tem uma dica de documentação?

Basicamente para ordenar alfabéticamente vc teria que normalizar a string. Isso significa passar tudo para caixa baixa ou alta e transformar caracteres especiais para normais. No seu caso, é -> e.

Para facilitar isso, vc poderia instalar um pacote que já faz isso. Por exemplo, esse para criar slugs visando urls semanticas:

<https://github.com/un33k/python-slugify> - Returns unicode slugs. Contribute to un33k/python-slugify development by creating an account on GitHub.

Dai então basta vc passar essa função aí como key da função sort:

**from slugify import slugify**

**sorted('Josué', key=slugify)**

### 1.4.10-Contagem de Caracteres com Dicionário

**Ctrl Shift J** junta as linhas no PyCharm

Com o mouse segurando **Alt** é possível ficar com multi-cursor

def contar\_caracteres(s):  
 *"""Função que conta os caracteres de uma string  
 Ex:  
  
 >>> contar\_caracteres('Marcelo')  
 {'M': 1, 'a': 1, 'c': 1, 'e': 1, 'l': 1, 'o': 1, 'r': 1}  
 >>> contar\_caracteres('banana')  
 {'a': 3, 'b': 1, 'n': 2}* ***:param*** *s: string a ser contada  
  
 """* resultado = {}  
  
 for caracter in s:  
 # contagem = resultado.get(caracter, 0)  
 # contagem += 1  
 # resultado[caracter] = contagem  
  
 # ou as três linhas acima em apenas uma

resultado[caracter] = resultado.get(caracter, 0) + 1

return resultado  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(contar\_caracteres('Marcelo'))  
 print()  
 print(contar\_caracteres('banana'))

### 1.4.11-Retrospectiva: Paradigma Procedural

# 2-Orientação a Objetos

Depois de aprender o paradigma procedural na seção anterior chega hora de conhecer outro: a Orientação a Objetos (OO). Você vai aprender sobre classes e seus componentes, herança e utilizar esses conceitos para implementar o jogo Python Birds. Como toda mudança de paradigma, demora um tempo para se acostumar, mas é importante aprender bem OO porque ela utilizada em inúmeras bibliotecas e frameworks.

**Capítulos**

Classe e Composição

Herança

## 2.1-Classe e Composição

Nesse capítulo você vai estudar e estrutura de dados base de toda Orientação a Objetos: a Classe. Além de serem abordados todos seu elementos, como atributos de dados e métodos, será também apresentado o conceito de Composição. Nele você aprenderá como os Objetos, construídos através de classes, se relacionam. Depois disso tudo você ainda aprender o básico sobre o controlador de versão Git e também sobre testes automáticos com Doctest.

2.1.1-Tópicos

2.1.2-Git no Windows

2.1.3-Git no Ubuntu

2.1.4-Git no Mac

2.1.5-Github e Setup

2.1.6-Classe

2.1.7-Commit e Push

2.1.8-Método

2.1.9-Atributo de Dado

2.1.10-Atributo Complexo

2.1.11-Atributo Dinâmico

2.1.12-Atributo de Classe

2.1.13-Método de Classe

2.1.14-Composição

2.1.15-Doctest

2.1.16-Implementação do Motor

2.1.17-Implementação da Direção

2.1.18-Implementação do Carro

2.1.19-Fase e Atores

### 2.1.1-Tópicos

### 2.1.2-Git no Windows

### 2.1.3-Git no Ubuntu

### 2.1.4-Git no Mac

### 2.1.5-Github e Setup

<https://github.com/pythonprobr/pythonbirds>

### 2.1.6-Classe

estrutura de dados base da Orientação a Objetos: a Classe

. novo pacote : OO > Add to Git (and always remember this decision)

. novo arquivo : pessoa > class – Segundo a PEP8 começando sua notação com Maiúscula (CamelCase)

>>> from oo.pessoa import Pessoa

>>> p = Pessoa() 🡸 não sabemos se é classe ou função... temos uma pista porque o nome de classes são em CamelCase; o nome de funções em snakecase

>>> type(p)

<class 'oo.pessoa.Pessoa'> 🡸 type indica o nome complete da função!

A classe é uma forma de criar seus tipos personalizados, como uma forma de gelo na qual – para produzir os objetos – você precisa botar água

### 2.1.7-Commit e Push

Nesse tópico você vai aprender a tirar uma foto do estado do seu código em determinado momento para manter todas mudanças em um histórico e enviar para seu repositório no Github. Depois dessa aula nunca mais você vai perder um arquivo.

### 2.1.8-Método

### 2.1.9-Atributo de Dado

### 2.1.10-Atributo Complexo

### 2.1.11-Atributo Dinâmico

### 2.1.12-Atributo de Classe

### 2.1.13-Método de Classe

### 2.1.14-Composição

### 2.1.15-Doctest

### 2.1.16-Implementação do Motor

### 2.1.17-Implementação da Direção

### 2.1.18-Implementação do Carro

### 2.1.19-Fase e Atores

## 2.2-Herança

2.1.1 -

2.1.2 -

2.1.3 -

2.1.4 -

2.1.5 -

2.1.6 -

2.1.7 -

2.1.8 -

2.1.9 -

2.1.10 -

2.1.11 -

2.1.12 -

2.1.13 -

2.1.14 -

2.1.15 -

2.1.16 -

2.1.17 -

2.1.18 -

2.1.19 -