Modulo2

Contents

1 Funções

1.1 Introducao

```
"""
Introducao as funcoes (def) em Python
Funcoes sao trechos de codigo usados para
replicar determinada acao ao longo do seu codigo.
Elas podem receber valores para parametros (argumentos)
e retornar um valor especifico.
Por padrao, funcoes Python retornam None (nada).
"""
# def Print(a, b, c):
#     print('Várias1')
#     print('Várias2')
#     print('Várias3')
#     print('Várias4')

# def imprimir(a, b, c):
#     print(a, b, c)


# imprimir(1, 2, 3)
# imprimir(4, 5, 6)

def saudacao(nome='Sem nome'):
    print(f'Olá, {nome}!')


saudacao('Luiz Otávio')
saudacao('Maria')
saudacao('Helena')
saudacao()
```

1.2 Argumentos

```
    """
Argumentos nomeados e não nomeados em funcoes Python
Argumento nomeado tem nome com sinal de igual
Argumento não nomeado recebe apenas o argumento (valor)
Parametro e passado no escopo da funcao, argumento quando vc chama a funcao.
"""


def soma(x, y, z):
    # Definição
    print(f'{x=} y={y} {z=}', '|', 'x + y + z = ', x + y + z)


soma(1, 2, 3)
soma(1, y=2, z=5)

print(1, 2, 3, sep='-')
```

1.3 Valores padrão para parametros

```
    """
Valores padrao para parametros
Ao definir uma funcao, os parametros podem
ter valores padrão. Caso o valor não seja
enviado para o parametro, o valor padrao será
usado.
Refatorar: editar o seu codigo.
"""
def soma(x, y, z=None):
    if z is not None:
        print(f'{x=} {y=} {z=}', x + y + z)
    else:
        print(f'{x=} {y=}', x + y)


soma(1, 2)
soma(3, 5)
soma(100, 200)
soma(7, 9, 0)
soma(y=9, z=0, x=7)
```

1.4 Escopo de funções

O global call stack -> pilha de chamadas

```
"""
Escopo de funcoes em Python
Escopo significa o local onde aquele codigo pode atingir.
Existe o escopo global e local.
O escopo global e o escopo onde todo o codigo e alcancavel.
O escopo local e o escopo onde apenas nomes do mesmo local
podem ser alcançados.
"""
x = 1
def escopo():
    global x
    x = 10
    def outra_funcao():
        global x
        x = 11
        y = 2
        print(x, y)
    outra_funcao()
    print(x)
print(x)
escopo()
print(x)
```

1.5 retorno dos valores das funções

```
"""
Retorno de valores das funcoes (return)
"""
def soma(x, y):
    if x > 10:
        return [10, 20]
    return x + y


# variavel = soma(1, 2)
# variavel = int('1')
soma1 = soma(2, 2)
soma2 = soma(3, 3)
print(soma1)
print(soma2)
print(soma(11, 55))
```

1.6 *args

```
    """
    args - Argumentos nao nomeados
    * - *args (empacotamento e desempacotamento)
    """
    # Lembre-te de desempacotamento
    # x, y, *resto = 1, 2, 3, 4
    # print(x, y, resto)


    # def soma(x, y):
    #     return x + y

    def soma(*args):
        total = 0
        for numero in args:
            total += numero
        return total


    soma_1_2_3 = soma(1, 2, 3)
    # print(soma_1_2_3)

    soma_4_5_6 = soma(4, 5, 6)
    # print(soma_4_5_6)

    numeros = 1, 2, 3, 4, 5, 6, 7, 78, 10
    outra_soma = soma(*numeros)
    print(outra_soma)

    print(sum(numeros))
    # print(*numeros)
```

1.7 Higher Order function

Termos técnicos: Higher Order Functions e First-Class Functions Academicamente, os termos Higher Order Functions e First-Class Functions têm significados diferentes.

Higher Order Functions -> Funções que podem receber e/ou retornar outras funções
First-Class Functions -> Funções que são tratadas como outros tipos de dados comuns (strings, inteiros, etc:.)
Não faria muita diferença no seu código, mas penso que deveria lhe informar isso.
Observação: esses termos podem ser diferentes e ainda refletir o mesmo significado.

```
"""
Higher Order Functions
Funcoes de primeira classe
"""


def saudacao(msg, nome):
    return f'{msg}, {nome}!'


def executa(funcao, *args):
    return funcao(*args)


print(
    executa(saudacao, 'Bom dia', 'Luiz')
)
print(
    executa(saudacao, 'Boa noite', 'Maria')
)
# def duplicar(numero):
#     return numero * 2
# def triplicar(numero):
#     return numero * 3
# def quadruplicar(numero):
#     return numero * 4

#multiplicador =  e o numero no qual vc deseja multiplicar
#e como se a funcao multiplicar passase a ser duplicar , quintuplicar etc.
def criar_multiplicador(multiplicador):
    def multiplicar(numero):
        return numero * multiplicador
    return multiplicar
duplicar = criar_multiplicador(2)
triplicar = criar_multiplicador(3)
quadruplicar = criar_multiplicador(4)
print(duplicar(2))
print(triplicar(2))
print(quadruplicar(2))
```

1.8 closure

```
    """
Closure e funcoes que retornam outras funcoes
"""
def criar_saudacao(saudacao):
    def saudar(nome):
        return f'{saudacao}, {nome}!'
    return saudar


falar_bom_dia = criar_saudacao('Bom dia')
falar_boa_noite = criar_saudacao('Boa noite')

for nome in ['Maria', 'Joana', 'Luiz']:
    print(falar_bom_dia(nome))
    print(falar_boa_noite(nome))
```

2 dict

2.1 Definições e conceitos

```
    # Dicionários em Python (tipo dict)
    # Dicionários são estruturas de dados do tipo
    # par de "chave" e "valor".
    # Chaves podem ser consideradas como o "indice"
    # que vimos na lista e podem ser de tipos imutáveis
    # como: str, int, float, bool, tuple, etc.
    # O valor pode ser de qualquer tipo, incluindo outro
    # dicionário.
    # Usamos as chaves - {} - ou a classe dict para criar
    # dicionários.
    # Imutáveis: str, int, float, bool, tuple
    # Mutável: dict, list
    # pessoa = {
```

```
    #     'nome': 'Luiz Otávio',
    #     'sobrenome': 'Miranda',
    #     'idade': 18,
    #     'altura': 1.8,
    #     'endereços': [
    #         {'rua': 'tal tal', 'numero': 123},
    #         {'rua': 'outra rua', 'numero': 321},
    #     ]
    # }
    # pessoa = dict(nome='Luiz Otávio', sobrenome='Miranda')
    pessoa = {
        'nome': 'Luiz Otávio',
        'sobrenome': 'Miranda',
        'idade': 18,
        'altura': 1.8,
        'endereços': [
            {'rua': 'tal tal', 'numero': 123},
            {'rua': 'outra rua', 'numero': 321},
        ],
    }
    # print(pessoa, type(pessoa))
    print(pessoa['nome'])
    print(pessoa['sobrenome'])

    print()

    for chave in pessoa:
        print(chave, pessoa[chave])
```

2.2 Manipulando chaves

```
    # Manipulando chaves e valores em dicionários
pessoa = {}
##
##
chave = 'nome'
pessoa[chave] = 'Luiz Otávio'
pessoa['sobrenome'] = 'Miranda'
print(pessoa[chave])
pessoa[chave] = 'Maria'
del pessoa['sobrenome']
print(pessoa)
print(pessoa['nome'])
# print(pessoa.get('sobrenome'))
if pessoa.get('sobrenome') is None:
    print('NAO EXISTE')
else:
    print(pessoa['sobrenome'])
# print('ISSO Não vai')
```

2.3 metodos uteis

```
    # Metodos uteis dos dicionarios em Python
    # len - quantas chaves
    # keys - iterável com as chaves
    # values - iterável com os valores
    # items - iterável com chaves e valores
    # setdefault - adiciona valor se a chave não existe

    pessoa = {
        'nome': 'Luiz Otávio',
        'sobrenome': 'Miranda',
        'idade': 900,
    }
    pessoa.setdefault('idade', 0)
    print(pessoa['idade'])
    # print(len(pessoa))
    # print(list(pessoa.keys()))
    # print(list(pessoa.values()))
    # print(list(pessoa.items()))

    # for valor in pessoa.values():
    #     print(valor)

    # for chave, valor in pessoa.items():
    #     print(chave, valor)
```

2.4 shallow copy vs deep copy

o shallow copy não copia valores mutaveis

```
# copy - retorna uma copia rasa (shallow copy)
import copy
d1 = {
    'c1': 1,
    'c2': 2,
    'l1': [0, 1, 2],
}
d2 = d1.copy()#copia rasa
d2 = copy.deepcopy(d1)#copia profunda
d2['c1'] = 1000
d2['l1'][1] = 999999
print(d1)
print(d2)
```

2.5 metodos uteis part2

```
# get - obtem uma chave
# pop - Apaga um item com a chave especificada (del)
# popitem - Apaga o ultimo item adicionado
# update - Atualiza um dicionário com outro
p1 = {
    'nome': 'Luiz',
    'sobrenome': 'Miranda',
}
# print(p1['nome'])
# print(p1.get('nome', 'Não existe'))

# nome = p1.pop('nome')
# print(nome)
# print(p1)
# ultima_chave = p1.popitem()
# print(ultima_chave)
# print(p1)
# p1.update({
#     'nome': 'novo valor',
#     'idade': 30,
# })
# p1.update(nome='novo valor', idade=30)
# tupla = (('nome', 'novo valor'), ('idade', 30))
lista = [['nome', 'novo valor'], ['idade', 30]]
p1.update(lista)
print(p1)
```

3 set

3.1 introdução ao tipo set

```
# Sets - Conjuntos em Python (tipo set)
# Conjuntos são ensinados na matemática
# https://brasilescola.uol.com.br/matematica/conjunto.htm
# Representados graficamente pelo diagrama de Venn
# Sets em Python sao mutaveis, porem aceitam apenas
# tipos imutáveis como valor interno.
# Criando um set
# set(iterável) ou {1, 2, 3}
# s1 = set('Luiz')
s1 = set()  # vazio
s1 = {'Luiz', 1, 2, 3}  # com dados
```

3.2 peculiaridades do tipo mutavel set

```
# Sets são eficientes para remover valores duplicados
# de iteráveis.
# - Não aceitam valores mutáveis;
# - Seus valores serao sempre unicos;
# - nao tem indexes;
# - não garantem ordem;
# - são iteráveis (for, in, not in)
# l1 = [1, 2, 3, 3, 3, 3, 3, 1]
# s1 = set(l1)
# l2 = list(s1)
# s1 = {1, 2, 3}
# print(3 not in s1)
# for numero in s1:
#     print(numero)
```

3.3 Metodos uteis

```
# Metodos uteis:
# add, update, clear, discard
# Operadores uteis:
# união | união (union) - Une
# intersecção & (intersection) - Itens presentes em ambos
# diferença - Itens presentes apenas no set da esquerda
# diferença simetrica ^ - Itens que nao estao em ambos
s1 = set()
s1.add('Luiz')
s1.add(1)
s1.update(('Olá mundo', 1, 2, 3, 4))#adiciona elementos ao set em questao
# s1.clear()
s1.discard('Olá mundo')#elima o valor do set em questão
s1.discard('Luiz')
print(s1)
```

3.4 operadores uteis

```
    # união | união (union) - Une
    # intersecção & (intersection) - Itens presentes em ambos
    # diferença - Itens presentes apenas no set da esquerda
    # diferença simetrica ^ - Itens que nao estao em ambos
    s1 = {1, 2, 3}
    s2 = {2, 3, 4}
    s3 = s1 | s2
    s3 = s1 & s2
    s3 = s2 - s1
    s3 = s1 ^ s2
    print(s3)
```

3.5 Exemplos de uso

```
    # Exemplo de uso dos sets
letras = set()
while True:
    letra = input('Digite: ')
    letras.add(letra.lower())

    if 'l' in letras:
        print('PARABENS')
        break

    print(letras)
```

4 lambda + sorted

```
# A função lambda e uma funçao como qualquer
# outra em Python. Porem, sao funcoes anonimas
# que contem apenas uma linha. Ou seja, tudo
# deve ser contido dentro de uma unica
# expressão.
# lista = [
#     {'nome': 'Luiz', 'sobrenome': 'miranda'},
#     {'nome': 'Maria', 'sobrenome': 'Oliveira'},
#     {'nome': 'Daniel', 'sobrenome': 'Silva'},
#     {'nome': 'Eduardo', 'sobrenome': 'Moreira'},
#     {'nome': 'Aline', 'sobrenome': 'Souza'},
# ]
# lista = [4, 32, 1, 34, 5, 6, 6, 21, ]
# lista.sort(reverse=True)
# sorted(lista)
lista = [
    {'nome': 'Luiz', 'sobrenome': 'miranda'},
    {'nome': 'Maria', 'sobrenome': 'Oliveira'},
    {'nome': 'Daniel', 'sobrenome': 'Silva'},
    {'nome': 'Eduardo', 'sobrenome': 'Moreira'},
    {'nome': 'Aline', 'sobrenome': 'Souza'},
]
def exibir(lista):
    for item in lista:
        print(item)
    print()
l1 = sorted(lista, key=lambda item: item['nome'])#ordenação por nome atraves do lambda
l2 = sorted(lista, key=lambda item: item['sobrenome'])#ordenção por sobrenome atraves do
    sobrenome.
exibir(l1)
exibir(l2)
```

4.1 funções lambdas com parametros complexos

```
    def executa(funcao, *args):
    return funcao(*args)

# def soma(x, y):
#     return x + y

# def cria_multiplicador(multiplicador):
#     def multiplica(numero):
#         return numero * multiplicador
#     return multiplica


# duplica = cria_multiplicador(2)
duplica = executa(
    lambda m: lambda n: n * m,
    2
)
print(duplica(2))

print(
    executa(
        lambda x, y: x + y,
        2, 3
    ),#passando uma expressão lmbda como parametro e, os valores
)

print(
    executa(
        lambda *args: sum(args),
        1, 2, 3, 4, 5, 6, 7
    )#soma todos  os valores passados como parametro
)
```

5 empacotamento e desmpacotamento em dicionarios

```
# Empacotamento e desempacotamento de dicionários
a, b = 1, 2
a, b = b, a
# print(a, b)
# (a1, a2), (b1, b2) = pessoa.items()#desempacota internamente o dict
# print(a1, a2)
# print(b1, b2)
# for chave, valor in pessoa.items():
#     print(chave, valor)
pessoa = {
    'nome': 'Aline',
    'sobrenome': 'Souza',
}
dados_pessoa = {
    'idade': 16,
    'altura': 1.6,
}
pessoas_completa = {**pessoa, **dados_pessoa}#união de dicionarios o "**" extrai os
   valores do dict
# print(pessoas_completa)
# args e kwargs
# args (já vimos)
# kwargs - keyword arguments (argumentos nomeados)
def mostro_argumentos_nomeados(*args, **kwargs):
    print('NAO NOMEADOS:', args)

    for chave, valor in kwargs.items():
        print(chave, valor)
# mostro_argumentos_nomeados(nome='Joana', qlq=123)
# mostro_argumentos_nomeados(**pessoas_completa)
configuracoes = {
    'arg1': 1,
    'arg2': 2,
    'arg3': 3,
    'arg4': 4,
}
mostro_argumentos_nomeados(**configuracoes)
```

6 list comprehension

```
# List comprehension e uma forma rapida para criar listas
# a partir de iteráveis.
```

```
# print(list(range(10)))
lista = []
for numero in range(10):
    lista.append(numero)
# print(lista)
lista = [
    numero * 2
    for numero in range(10)
]
print(lista)
```

6.1 Mapeamento de dados em list comprehension

```
    produtos = [
        {'nome': 'p1', 'preco': 20, },
        {'nome': 'p2', 'preco': 10, },
        {'nome': 'p3', 'preco': 30, },
    ]
    novos_produtos = [
        {**produto, 'preco': produto['preco'] * 1.05}
        if produto['preco'] > 20 else {**produto}
        for produto in produtos
    ]

    # print(novos_produtos)
    print(*novos_produtos, sep='\n')

    print com pprint
    import pprint
    pprint.pprint(novos_produtos)
```

6.2 filtro em list comprehension

```
    # a partir de iteráveis.
    # print(list(range(10)))
    import pprint
    def p(v):
        pprint.pprint(v, sort_dicts=False, width=40)#printa o dict de forma mais
            organizada


    lista = []
    for numero in range(10):
        lista.append(numero)
    # print(lista)

    lista = [
        numero * 2
        for numero in range(10)
    ]
    # print(list(range(10)))
    # print(lista)


    # Mapeamento de dados em list comprehension
    produtos = [
        {'nome': 'p1', 'preco': 20, },
        {'nome': 'p2', 'preco': 10, },
        {'nome': 'p3', 'preco': 30, },
    ]
    novos_produtos = [
        {**produto, 'preco': produto['preco'] * 1.05}
        if produto['preco'] > 20 else {**produto}
        for produto in produtos
    ]

    # # print(novos_produtos)
    # print(novos_produtos)
    # p(novos_produtos)
    # lista = [n for n in range(10) if n < 5]
    novos_produtos = [
        {**produto, 'preco': produto['preco'] * 1.05}
        if produto['preco'] > 20 else {**produto}
        for produto in produtos
        if (produto['preco'] >= 20 and produto['preco'] * 1.05) > 10
    ]# filter
    p(novos_produtos)
```

6.3 list comprehension: for + for

```
    lista = []
for x in range(3):
    for y in range(3):
        lista.append((x, y))
lista = [
    (x, y)
    for x in range(3)
    for y in range(3)
]
lista = [
    [(x, letra) for letra in 'Luiz']
    for x in range(3)
]

print(lista)
```

7 dictionary comprehension set comprehension

```
    # Dictionary Comprehension e Set Comprehension
produto = {
    'nome': 'Caneta Azul',
    'preco': 2.5,
    'categoria': 'Escritorio',
}

dc = {
    chave: valor.upper()
    if isinstance(valor, str) else valor
    for chave, valor
    in produto.items()
    if chave != 'categoria'
}

lista = [
    ('a', 'valor a'),
    ('b', 'valor a'),
    ('b', 'valor a'),
]
dc = {
    chave: valor
    for chave, valor in lista
}

s1 = {2 ** i for i in range(10)}#converte lista em dict
print(s1)
```

8 isinstance

```
# isinstace  para saber se objeto e de determinado tipo
lista = [
    'a', 1, 1.1, True, [0, 1, 2], (1, 2),
    {0, 1}, {'nome': 'Luiz'},
]

for item in lista:
    if isinstance(item, set):
        print('SET')
        item.add(5)
        print(item, isinstance(item, set))

    elif isinstance(item, str):
        print('STR')
        print(item.upper())

    elif isinstance(item, (int, float)):
        print('NUM')
        print(item, item * 2)
    else:
        print('OUTRO')
        print(item)
```

9 Valores Truthy e Falsy, Tipos Mutáveis e Imutáveis

```
# Mutáveis [] {} set()
# Imutáveis () "" 0 0.0 None False range(0, 10)
lista = []
dicionario = {}
```

```
conjunto = set()
tupla = ()
string = ''
inteito = 0
flutuante = 0.0
nada = None
falso = False
intervalo = range(0)
def falsy(valor):
    return 'falsy'if not valor else 'truthy'
print(f'TESTE', falsy('TESTE'))
print(f'{lista=}', falsy(lista))
print(f'{dicionario=}', falsy(dicionario))
print(f'{conjunto=}', falsy(conjunto))
print(f'{tupla=}', falsy(tupla))
print(f'{string=}', falsy(string))
print(f'{inteito=}', falsy(inteito))
print(f'{flutuante=}', falsy(flutuante))
print(f'{nada=}', falsy(nada))
print(f'{falso=}', falsy(falso))
print(f'{intervalo=}', falsy(intervalo))
```

10 dir, hasattr e getattr em Python

```
string = 'Luiz'
metodo = 'strip'
#hasattr checa se o objeto tem o metodo especificado disponivel
if hasattr(string, metodo):
    print('Existe upper')
    print(getattr(string, metodo)())#usa metodo especificado no condicional
else:
    print('Nao existe o metodo', metodo)
```

11 Generator Expressions

11.1 iterable e iterators

```
    iterable = ['Eu', 'Tenho', '__iter__']
    iterator = iter(iterable)# tem _iter__ e next
    iterator = __iter__.iterable
    print(next((iterator)))
    print(next((iterator)))
```

11.2 Generator

```
import sys
# Generator expression, Iterables e Iterators em Python
iterable = ['Eu', 'Tenho', '__iter__']
iterator = iter(iterable)  # tem __iter__ e __next__
lista = [n for n in range(1000000)]
generator = (n for n in range(1000000))#espera pelo chamado da função Generator portanto
    não esta salvo na memoria , como a list
print(sys.getsizeof(lista)) #verifica o tamanho da lista
print(sys.getsizeof(generator))
print(generator)

# for n in generator:
#     print(n)
```

11.3 Introducao a generator functions

```
# generator = (n for n in range(1000000))
def generator(n=0, maximum=10):
    while True:
        yield n # a função pausa aqui
        n += 1

        if n >= maximum:
            return
gen = generator(maximum=1000000)
#porem para acessar a yield e necessario um iterador
for n in gen:
    print(n)
```

11.4 yield from

```
    # Yield from
def gen1():
    print('COMECOU GEN1')
    yield 1
    yield 2
    yield 3
    print('ACABOU GEN1')

def gen3():
    print('COMECOU GEN3')
    yield 10
    yield 20
    yield 30
    print('ACABOU GEN3')

def gen2(gen=None):
    print('COMECOU GEN2')
    if gen is not None:
        yield from gen
    yield 4
    yield 5
    yield 6
    print('ACABOU GEN2')

g1 = gen2(gen1())
g2 = gen2(gen3())
g3 = gen2()
for numero in g1:
    print(numero)
print()
for numero in g2:
    print(numero)
print()
for numero in g3:
    print(numero)
print()
```

12 Try Except else finally

12.1 Introdução

```
# a = 18
# b = 0
# c = a / b
try:
    a = 18
    b = 0
    # print(b[0])
    print('Linha 1'[1000])
    c = a / b
    print('Linha 2')
except ZeroDivisionError:
    print('Dividiu por zero.')
except NameError:
    print('Nome b não está definido')
except (TypeError, IndexError):
    print('TypeError + IndexError')
except Exception:
    print('ERRO DESCONHECIDO.')
print('CONTINUAR')
```

12.2 Localizando tipo especifico error

```
    # a = 18
    # b = 0
    # c = a / b

    try:
        a = 18
        b = 0
        # print(b[0])
        # print('Linha 1'[1000])
        c = a / b
        print('Linha 2')
    except ZeroDivisionError as e:
        print(e.__class__.__name__)
        print(e)
```

```
    except NameError:
        print('Nome b não está definido')
    except (TypeError, IndexError) as error:#cria variavel com o nome do erro
        print('TypeError + IndexError')
        print('MSG:', error)
        print('Nome:', error.__class__.__name__)#na classe error pega o erro especifico
    except Exception:
        print('ERRO DESCONHECIDO.')

    print('CONTINUAR')
```

12.3 try, except, built-in

Link documentação dos errors disponiveis em python

```
    # try, except, else e finally
    #pode ser usado so com o finally que não sera tratada a excecao
    try:
        print('ABRIR ARQUIVO')
        8/0
    except ZeroDivisionError as e:
        print(e.__class__.__name__)
        print(e)
        print('DIVIDIU ZERO')
    except IndexError as error:#captura o erro com uma variavel
        print('IndexError')
    except (NameError, ImportError):
        print('NameError, ImportError')
    else:
        print('Não deu erro')
    finally:
        print('FECHAR ARQUIVO')
```

12.4 raise

Link documentação dos errors disponiveis em python

```
    # raise - lançando excecoes (erros)
def nao_aceito_zero(d):
    if d == 0:
        raise ZeroDivisionError('Voce esta tentando dividir por zero')#criando erro
    return True
def deve_ser_int_ou_float(n):
    tipo_n = type(n)
    if not isinstance(n, (float, int)):
        raise TypeError(
            f'"{n}" deve ser int ou float. '
            f'"{tipo_n.__name__}" enviado.'
        )
    return True


def divide(n, d):
    deve_ser_int_ou_float(n)
    deve_ser_int_ou_float(d)
    nao_aceito_zero(d)
    return n / d


print(divide(8, '0'))
```

13 Módulos padrão do Python (import, from, as e *)

13.1 Introdução

Link documentação modulo python

```
# inteiro - import nome_modulo
# Vantagens: voce tem o namespace do modulo
# Desvantagens: nomes grandes
# import sys
# platform = 'A MINHA'
# print(sys.platform)
# print(platform)

# partes - from nome_modulo import objeto1, objeto2
# Vantagens: nomes pequenos
# Desvantagens: Sem o namespace do modulo
# from sys import exit, platform

# print(platform)#tomar cuidado pra nao subescrever a variavel chamada no import
```

```
# alias 1 - import nome_modulo as apelido
# import sys as s

# sys = 'alguma coisa'
# print(s.platform)
# print(sys)


# alias 2 - from nome_modulo import objeto as apelido
# from sys import exit as ex
# from sys import platform as pf #apelidando as funcoes tiradas do modulo

# print(pf)

# Vantagens: voce pode reservar nomes para seu codigo
# Desvantagens: pode ficar fora do padrão da linguagem

# má prática - from nome_modulo import *
# Vantagens: importa tudo de um modulo
# Desvantagens: importa tudo de um modulo
# from sys import exit, platform

# print(platform)
# exit()
```

13.2 Modularização

```
    # Modularizacao - Entendendo os seus proprios modulos Python
    # O primeiro modulo executado chama-se __main__
    # Voce pode importar outro modulo inteiro ou parte do modulo
    # O python conhece a pasta onde o __main__ está e as pastas
    # abaixo dele.
    # Ele nao reconhece pastas e modulos acima do __main__ por
    # padrão
    # O python conhece todos os modulos e pacotes presentes
    # nos caminhos de sys.path
    #o python so reconhece modulos para tras dele
    #aula97
    import aula97_m
    try:
        import sys
        sys.path.append('/home')#incluindo um nvo caminho na lista , voce importa o
           modulo do diretorio especificado

    except: ModuleNotFoundError:
        ...
    print('Este modulo se chama', __name__)
    #aula97_m
    print('Este modulo se chama', __name__)
```

13.3 Recarregando modulos, importlib, e singleton

singleton significa que só pode haver uma isntancia desse modulo.

```
#aula_98
import importlib
import aula98_m
print(aula98_m.variavel)
for i in range(10):
    importlib.reload(aula98_m)#o modulo so e recarregado caso seja solicitado
    print(i)
print('Fim')
#aula 98_m
print(123)
variavel = 'Luiz 1'
```

14 Packages

```
    #aula99
from sys import path
import aula99_package.modulo
from aula99_package import modulo
from aula99_package.modulo import *#conectado ao __all__
# from aula99_package.modulo import soma_do_modulo

# print(*path, sep='\n')
print(soma_do_modulo(1, 2))
print(aula99_package.modulo.soma_do_modulo(1, 2))
print(modulo.soma_do_modulo(1, 2))
print(variavel)
```

```
print(nova_variavel)

    #aula99_m
# __all__ = [
    #     'variavel',
    #     'soma_do_modulo',
    #     'nova_variavel',
    # ]#lista que contem o que sera importado.

    variavel = 'Alguma coisa'


    def soma_do_modulo(x, y):
        return x + y


    nova_variavel = 'OK'
```

14.1 observações

```
    "[python]": {
    "editor.defaultFormatter": "ms-python.python",
    "editor.tabSize": 4,
    "editor.insertSpaces": true,
    "editor.insertSpaces": false,
    "editor.formatOnSave": true
    // "editor.codeActionsOnSave": {
    //   "source.fixAll": true,
    }}
    @@ -1,14 +1,18 @@
from sys import path
# from sys import path

import aula99_package.modulo
from aula99_package import modulo
from aula99_package.modulo import *
# import aula99_package.modulo
# from aula99_package import modulo
# from aula99_package.modulo import *

# from aula99_package.modulo import soma_do_modulo
# # from aula99_package.modulo import soma_do_modulo

# print(*path, sep='\n')
print(soma_do_modulo(1, 2))
print(aula99_package.modulo.soma_do_modulo(1, 2))
print(modulo.soma_do_modulo(1, 2))
print(variavel)
print(nova_variavel)
# # print(*path, sep='\n')
# print(soma_do_modulo(1, 2))
# print(aula99_package.modulo.soma_do_modulo(1, 2))
# print(modulo.soma_do_modulo(1, 2))
# print(variavel)
# print(nova_variavel)
from aula99_package.modulo import fala_oi, soma_do_modulo

print(__name__)
fala_oi()
```

14.2 init.py

```
    # import aula99_package.modulo
    # from aula99_package import modulo

    # print(modulo.soma_do_modulo(1, 2))
    # print(variavel)
    # print(nova_variavel)
    from aula99_package.modulo import fala_oi, soma_do_modulo
    # from aula99_package.modulo import fala_oi, soma_do_modulo

    print(__name__)
    fala_oi()
    # print(__name__)
    # fala_oi()

    from aula99_package import falar_oi, soma_do_modulo

    print(soma_do_modulo(2, 3))
    falar_oi()
```

```
    from aula99_package.modulo import *
    from aula99_package.modulo_b import *

from aula99_package.modulo_b import fala_oi
# from aula99_package.modulo_b import fala_oi
variavel = 'Alguma coisa'

def fala_oi():
def falar_oi():
print('oi')
```

15 Variaveis livres

```
    # Variáveis livres + nonlocal (locals, globals)
# print(globals())
# def fora(x):
#     a = x #variavel livre porque nao esta definida dentro do escopo da função dentro

#     def dentro():
#         # print(locals())

#         return a
#     return dentro


# dentro1 = fora(10)
# dentro2 = fora(20)

# print(dentro1())
# print(dentro2())
def concatenar(string_inicial):
    valor_final = string_inicial

    def interna(valor_a_concatenar=''):
        nonlocal valor_final# puxa a variavel para este escopo
        valor_final += valor_a_concatenar
        return valor_final
    return interna


c = concatenar('a')
print(c('b'))
print(c('c'))
print(c('d'))
final = c()
print(final)
```

16 Funções decoradoras

16.1 introdução

```
    # Funcoes decoradoras e decoradores
# Decorar = Adicionar / Remover/ Restringir / Alterar
# Funcoes decoradoras sao funcoes que decoram outras funcoes
# Decoradores são usados para fazer o Python
# usar as funcoes decoradoras em outras funcoes.

def criar_funcao(func):
    def interna(*args, **kwargs):
        print('Vou te decorar')
        for arg in args:
            e_string(arg)
        resultado = func(*args, **kwargs)
        print(f'O seu resultado foi {resultado}.')
        print('Ok, agora voce foi decorada')
        return resultado
    return interna


def inverte_string(string):
    return string[::-1]


def e_string(param):
    if not isinstance(param, str):
        raise TypeError('param deve ser uma string')
```

```
inverte_string_checando_parametro = criar_funcao(inverte_string)
invertida = inverte_string_checando_parametro('123')
print(invertida)
```

16.2 Decoradores em Python

```
# Funcoes decoradoras e decoradores
# Decorar = Adicionar / Remover/ Restringir / Alterar
# Funcoes decoradoras são funcoes que decoram outras funcoes
# Decoradores são usados para fazer o Python
# usar as funcoes decoradoras em outras funcoes.
# Decoradores são "Syntax Sugar" (Acucar sintatico)

def criar_funcao(func):
    def interna(*args, **kwargs):
        print('Vou te decorar')
        for arg in args:
            e_string(arg)
        resultado = func(*args, **kwargs)
        print(f'O seu resultado foi {resultado}.')
        print('Ok, agora voce foi decorada')
        return resultado
    return interna


@criar_funcao
def inverte_string(string):
    print(f'{inverte_string.__name__}')#checa nome da função no decorador
    return string[::-1]


def e_string(param):
    if not isinstance(param, str):
        raise TypeError('param deve ser uma string')


invertida = inverte_string('123')
print(invertida)
```

16.3 Decoradores com parametros

```
Decoradores com parametros
def fabrica_de_decoradores(a=None, b=None, c=None):
    def fabrica_de_funcoes(func):
        print('Decoradora 1')

        def aninhada(*args, **kwargs):
            print('Parametros do decorador, ', a, b, c)
            print('Aninhada')
            res = func(*args, **kwargs)
            return res
        return aninhada
    return fabrica_de_funcoes


@fabrica_de_decoradores(1, 2, 3)
def soma(x, y):
    return x + y


decoradora = fabrica_de_decoradores()
multiplica = decoradora(lambda x, y: x * y)

dez_mais_cinco = soma(10, 5)
dez_vezes_cinco = multiplica(10, 5)
print(dez_mais_cinco)
print(dez_vezes_cinco)
```

16.4 Ordem de aplicação de decoradores

```
def parametros_decorador(nome):
def decorador(func):
    print('Decorador:', nome)

    def sua_nova_funcao(*args, **kwargs):
        res = func(*args, **kwargs)
        final = f'{res} {nome}'
        return final
    return sua_nova_funcao
```

```
    return decorador


@parametros_decorador(nome='5')
@parametros_decorador(nome='4')
@parametros_decorador(nome='3')
@parametros_decorador(nome='2')
@parametros_decorador(nome='1')#ele aplica por ordem
def soma(x, y):
    return x + y


dez_mais_cinco = soma(10, 5)
print(dez_mais_cinco)
```

17 count é um iterador sem fim

```
# count e um iterador sem fim (itertools)
from itertools import count

c1 = count(step=8, start=8)
r1 = range(8, 100, 8)

print('c1', hasattr(c1, '__iter__'))
print('c1', hasattr(c1, '__next__'))
print('r1', hasattr(r1, '__iter__'))
print('r1', hasattr(r1, '__next__'))

print('count')
for i in c1:
    if i >= 100:
        break

    print(i)
print()
print('range')
for i in r1:
    print(i)
```

18 Combinations, Permutations, e product - itertools

```
    Combinations, Permutations e Product - Itertools
    # Combinação - Ordem não importa - iterável + tamanho do grupo
    # Permutação - Ordem importa
    # Produto - Ordem importa e repete valores unicos
    from itertools import combinations, permutations, product


    def print_iter(iterator):
        print(*list(iterator), sep='\n')
        print()


    pessoas = [
        'João', 'Joana', 'Luiz', 'Leticia',
    ]
    camisetas = [
        ['preta', 'branca'],
        ['p', 'm', 'g'],
        ['masculino', 'feminino', 'unisex'],
        ['algodão', 'poliester']
    ]

    print_iter(combinations(pessoas, 2))
    print_iter(permutations(pessoas, 2))
    print_iter(product(*camisetas))
```

19 groupby

```
    from itertools import groupby

alunos = [
    {'nome': 'Luiz', 'nota': 'A'},
    {'nome': 'Leticia', 'nota': 'B'},
    {'nome': 'Fabricio', 'nota': 'A'},
    {'nome': 'Rosemary', 'nota': 'C'},
    {'nome': 'Joana', 'nota': 'D'},
```

```
    {'nome': 'João', 'nota': 'A'},
    {'nome': 'Eduardo', 'nota': 'B'},
    {'nome': 'Andre', 'nota': 'A'},
    {'nome': 'Anderson', 'nota': 'C'},
]


def ordena(aluno):
    return aluno['nota']


alunos_agrupados = sorted(alunos, key=ordena)
grupos = groupby(alunos_agrupados, key=ordena)

for chave, grupo in grupos:
    print(chave)
    for aluno in grupo:
        print(aluno)
```

20 map, partial, GeneratorType, esgotamento de iterator

```
from functools import partial
from types import GeneratorType
# map - para mapear dados
def print_iter(iterator):
    print(*list(iterator), sep='\n')
    print()


produtos = [
    {'nome': 'Produto 5', 'preco': 10.00},
    {'nome': 'Produto 1', 'preco': 22.32},
    {'nome': 'Produto 3', 'preco': 10.11},
    {'nome': 'Produto 2', 'preco': 105.87},
    {'nome': 'Produto 4', 'preco': 69.90},
]


def aumentar_porcentagem(valor, porcentagem):
    return round(valor * porcentagem, 2)


aumentar_dez_porcento = partial(
    aumentar_porcentagem,
    porcentagem=1.1
)

# novos_produtos = [
#     {**p,
#         'preco': aumentar_dez_porcento(p['preco'])}
#     for p in produtos
# ]
def muda_preco_de_produtos(produto):
    return {
        **produto,
        'preco': aumentar_dez_porcento(
            produto['preco']
        )
    }
novos_produtos = list(map(
    muda_preco_de_produtos,
    produtos
))


print_iter(produtos)
print_iter(novos_produtos)

print(
    list(map(
        lambda x: x * 3,
        [1, 2, 3, 4]
    ))
)
```

21 filter

```
#filter e um filtro funcional
def print_iter(iterator):
```

```
    print(*list(iterator), sep='\n')
    print()


produtos = [
    {'nome': 'Produto 5', 'preco': 10.00},
    {'nome': 'Produto 1', 'preco': 22.32},
    {'nome': 'Produto 3', 'preco': 10.11},
    {'nome': 'Produto 2', 'preco': 105.87},
    {'nome': 'Produto 4', 'preco': 69.90},
]
def filtrar_preco(produto):
    return produto['preco'] > 100


# novos_produtos = [
#     p for p in produtos
#     if p['preco'] > 100
# ]
novos_produtos = filter(
    # lambda produto: produto['preco'] > 100,
    filtrar_preco,
    produtos
)
print_iter(produtos)
print_iter(novos_produtos)
```

22 Reduce

reduce faz a redução de um iterável em um valor

```
from functools import reduce

produtos = [
    {'nome': 'Produto 5', 'preco': 10},
    {'nome': 'Produto 1', 'preco': 22},
    {'nome': 'Produto 3', 'preco': 2},
    {'nome': 'Produto 2', 'preco': 6},
    {'nome': 'Produto 4', 'preco': 4},
]


# def funcao_do_reduce(acumulador, produto):
#     print('acumulador', acumulador)
#     print('produto', produto)
#     print()
#     return acumulador + produto['preco']


total = reduce(
    lambda ac, p: ac + p['preco'],
    produtos,
    0
)

print('Total e', total)


# total = 0
# for p in produtos:
#     total += p['preco']

# print(total)

# print(sum([p['preco'] for p in produtos]))
1 com
```

23 Funções recursivas

introdução

```
# Funcoes recursivas e recursividade
# - funcoes que podem se chamar de volta
# - uteis p/ dividir problemas grandes em partes menores
# Toda função recursiva deve ter:
# - Um problema que possa ser dividido em partes menores
# - Um caso recursivo que resolve o pequeno problema
# - Um caso base que para a recursão
# - fatorial - n! = 5! = 5 * 4 * 3 * 2 * 1 = 120
# https://brasilescola.uol.com.br/matematica/fatorial.htm
```

```python
def recursiva(inicio=0, fim=4):

    print(inicio, fim)

    # Caso base
    if inicio >= fim:
        return fim

    # Caso recursivo
    # contar ate chegar ao final
    inicio += 1
    return recursiva(inicio, fim)

print(recursiva())
```

23.1 Limites de recursividade

```python
    # - Um caso base que para a recursão
# - fatorial - n! = 5! = 5 * 4 * 3 * 2 * 1 = 120
# https://brasilescola.uol.com.br/matematica/fatorial.htm
def recursiva(inicio=0, fim=4):
# import sys

    print(inicio, fim)
# sys.setrecursionlimit(1004)

    # Caso base
    if inicio >= fim:
        return fim

    # Caso recursivo
    # contar ate chegar ao final
    inicio += 1
    return recursiva(inicio, fim)
# def recursiva(inicio=0, fim=4):

#     print(inicio, fim)

print(recursiva())
#     # Caso base
#     if inicio >= fim:
#         return fim

#     # Caso recursivo
#     # contar ate chegar ao final
#     inicio += 1
#     return recursiva(inicio, fim)


# print(recursiva(0, 1001))

def factorial(n):
    if n <= 1:
        return 1

    return n * factorial(n - 1)


print(factorial(5))
print(factorial(10))
print(factorial(100))
```

24 Ambientes virtuais

24.1 Introdução

```python
# Ambientes virtuais em Python (venv)
# Um ambiente virtual carrega toda a sua instalação
# do Python para uma pasta no caminho escolhido.
# Ao ativar um ambiente virtual, a instalação do
# ambiente virtual será usada.
# venv e o modulo que vamos usar para
# criar ambientes virtuais.
# Voce pode dar o nome que preferir para um
# ambiente virtual, mas os mais comuns são:
# venv env .venv .env
```

25 criando arquivos com python + context manager with

```
    Criando arquivos com Python + Context Manager with
    # Usamos a função open para abrir
    # um arquivo em Python (ele pode ou não existir)
    # Modos:
    # r (leitura), w (escrita), x (para criação)
    # a (escreve ao final), b (binário)
    # t (modo texto), + (leitura e escrita)
    # Context manager - with (abre e fecha)
    # Metodos uteis
    # write, read (escrever e ler)
    # writelines (escrever várias linhas)
    # seek (move o cursor)
    # readline (ler linha)
    # readlines (ler linhas)
    # Vamos falar mais sobre o modulo os, mas:
    # os.remove ou unlink - apaga o arquivo
    # os.rename - troca o nome ou move o arquivo
    # Vamos falar mais sobre o modulo json, mas:
    # json.dump = Gera um arquivo json
    # json.load
    caminho_arquivo = 'aula116.txt'

    # arquivo = open(caminho_arquivo, 'w')
    # #
    # arquivo.close()
    with open(caminho_arquivo, 'w') as arquivo:
        print('Olá mundo')
        print('Arquivo vai ser fechado')
```

25.1 Escrevendo no Arquivo

```
    Criando arquivos com Python + Context Manager with
    # with open (context manager) e metodos uteis do TextIOWrapper
    # Usamos a função open para abrir
    # um arquivo em Python (ele pode ou não existir)
    # Modos:
    @@ -23,6 +23,28 @@
    # arquivo = open(caminho_arquivo, 'w')
    # #
    # arquivo.close()
    with open(caminho_arquivo, 'w') as arquivo:
        print('Olá mundo')
        print('Arquivo vai ser fechado')

    with open(caminho_arquivo, 'w+') as arquivo:
        arquivo.write('Linha 1\n')
        arquivo.write('Linha 2\n')
        arquivo.writelines(
            ('Linha 3\n', 'Linha 4\n')
        )
        arquivo.seek(0, 0)
        print(arquivo.read())
        print('Lendo')
        arquivo.seek(0, 0)
        print(arquivo.readline(), end='')
        print(arquivo.readline().strip())
        print(arquivo.readline().strip())

        print('READLINES')
        arquivo.seek(0, 0)
        for linha in arquivo.readlines():
            print(linha.strip())


    print('#' * 10)

    with open(caminho_arquivo, 'r') as arquivo:
        print(arquivo.read())
```

25.2 Modos de abertura de arquivos

```
    # Leia tambem: https://www.otaviomiranda.com.br/2020/normalizacao-unicode-em-python/

# Usamos a função open para abrir
# um arquivo em Python (ele pode ou não existir)
@@ -24,27 +25,35 @@
# #
# arquivo.close()

with open(caminho_arquivo, 'w+') as arquivo:
```

```
# with open(caminho_arquivo, 'w+') as arquivo:
#     arquivo.write('Linha 1\n')
#     arquivo.write('Linha 2\n')
#     arquivo.writelines(
#         ('Linha 3\n', 'Linha 4\n')
#     )
#     arquivo.seek(0, 0)
#     print(arquivo.read())
#     print('Lendo')
#     arquivo.seek(0, 0)
#     print(arquivo.readline(), end='')
#     print(arquivo.readline().strip())
#     print(arquivo.readline().strip())

#     print('READLINES')
#     arquivo.seek(0, 0)
#     for linha in arquivo.readlines():
#         print(linha.strip())


# print('#' * 10)

# with open(caminho_arquivo, 'r') as arquivo:
#     print(arquivo.read())

with open(caminho_arquivo, 'w', encoding='utf8') as arquivo:
    arquivo.write('Atenção\n')
    arquivo.write('Linha 1\n')
    arquivo.write('Linha 2\n')
    arquivo.writelines(
        ('Linha 3\n', 'Linha 4\n')
    )
    arquivo.seek(0, 0)
    print(arquivo.read())
    print('Lendo')
    arquivo.seek(0, 0)
    print(arquivo.readline(), end='')
    print(arquivo.readline().strip())
    print(arquivo.readline().strip())

    print('READLINES')
    arquivo.seek(0, 0)
    for linha in arquivo.readlines():
        print(linha.strip())


print('#' * 10)

with open(caminho_arquivo, 'r') as arquivo:
    print(arquivo.read())
```

26 Módulo os

```
 import os
# Vamos falar mais sobre o modulo os, mas:
# os.remove ou unlink - apaga o arquivo
# os.rename - troca o nome ou move o arquivo
# Vamos falar mais sobre o modulo json, mas:
# json.dump = Gera um arquivo json
# json.load
caminho_arquivo = 'aula116.txt'

# arquivo = open(caminho_arquivo, 'w')
# #
# arquivo.close()

# with open(caminho_arquivo, 'w+') as arquivo:
#     arquivo.write('Linha 1\n')
#     arquivo.write('Linha 2\n')
#     arquivo.writelines(
#         ('Linha 3\n', 'Linha 4\n')
#     )
#     arquivo.seek(0, 0)
#     print(arquivo.read())
#     print('Lendo')
#     arquivo.seek(0, 0)
#     print(arquivo.readline(), end='')
#     print(arquivo.readline().strip())
#     print(arquivo.readline().strip())
```

```
#     print('READLINES')
#     arquivo.seek(0, 0)
#     for linha in arquivo.readlines():
#         print(linha.strip())


# print('#' * 10)

# with open(caminho_arquivo, 'r') as arquivo:
#     print(arquivo.read())

with open(caminho_arquivo, 'w', encoding='utf8') as arquivo:
    arquivo.write('Atenção\n')
    arquivo.write('Linha 1\n')
    arquivo.write('Linha 2\n')
    arquivo.writelines(
        ('Linha 3\n', 'Linha 4\n')
    )

# os.remove(caminho_arquivo) # ou unlink
# os.rename(caminho_arquivo, 'aula116-2.txt')
```

27 JSON

27.1 exemplo json e funções auxiliares

```
    import json
import os
"""
pessoas = [
    {
        "nome": 'maria',
        "sobrenome": " viera"
    },
    {
        "nome": "marcos",
        "sobrenome":"antonio"
    }


]
BASE_DIR =  os.path.dirname(__file__)
SAVE_TO = os.path.join(BASE_DIR, 'arquivo-python.json')

with open(SAVE_TO, 'w') as file:
    json.dump(pessoas, file, indent = 2) #pega os dados dentro do codigo
    print (json.dumps(pessoas, indent = 2))

# carregando aqruivo de fora
BASE_DIR = os.path.dirname(__file__)
JSON_FILE =  os.path.join(BASE_DIR, 'arquivo-python.json')

with open(JSON_FILE, 'r') as file:
    pessoas= json.load(file)# carreaga dados externos ao codigo
    print(json.dumps(pessoas))

    #for pessoa in pessoas:
     #   print(pessoa["nome"])"""
json_string = '''
[{"nome": "maria", "sobrenome": " viera"}, {"nome": "marcos", "sobrenome": "antonio"}]
'''
pessoas = json.loads(json_string)
for pessoa in pessoas:
    print(pessoa["nome"])
```

27.2 utilizando json em arquivos com python

```
    {
  "nome": "Luiz Otávio 2",
  "sobrenome": "Miranda",
  "enderecos": [
    {
      "rua": "R1",
      "numero": 32
    },
    {
      "rua": "R2",
      "numero": 55
    }
  ],
```

```
  "altura": 1.8,
  "numeros_preferidos": [
    2,
    4,
    6,
    8,
    10
  ],
  "dev": true,
  "nada": null
}
```

```
import json

pessoa = {
    'nome': 'Luiz Otávio 2',
    'sobrenome': 'Miranda',
    'enderecos': [
        {'rua': 'R1', 'numero': 32},
        {'rua': 'R2', 'numero': 55},
    ],
    'altura': 1.8,
    'numeros_preferidos': (2, 4, 6, 8, 10),
    'dev': True,
    'nada': None,
}
# encoding codifica o dict para a linguagem em uso
with open('aula117.json', 'w', encoding='utf8') as arquivo:
    json.dump(
        pessoa,
        arquivo,
        ensure_ascii=False,
        indent=2,
    )
# obtem os dados de fora contidos em um json
with open('aula117.json', 'r', encoding='utf8') as arquivo:
    pessoa = json.load(arquivo)
    # print(pessoa)
    # print(type(pessoa))# converte a estrutura de dados
    print(pessoa['nome'])
```

28 Problema dos parâmetros mutáveis em funções Python

```
def adiciona_clientes(nome, lista=None):
    if lista is None:
        lista = []
    lista.append(nome)
    return lista


cliente1 = adiciona_clientes('luiz')
adiciona_clientes('Joana', cliente1)
adiciona_clientes('Fernando', cliente1)
cliente1.append('Edu')

cliente2 = adiciona_clientes('Helena')
adiciona_clientes('Maria', cliente2)

cliente3 = adiciona_clientes('Moreira')
adiciona_clientes('Vivi', cliente3)

print(cliente1)
print(cliente2)
print(cliente3)
```

29 evitando o uso de condicionais

```
# Exercicio - Lista de tarefas com desfazer e refazer
# Musica para codar =)
# Everybody wants to rule the world - Tears for fears
# todo = [] -> lista de tarefas
# todo = ['fazer cafe'] -> Adicionar fazer cafe
# todo = ['fazer cafe', 'caminhar'] -> Adicionar caminhar
# desfazer = ['fazer cafe',] -> Refazer ['caminhar']
# desfazer = [] -> Refazer ['caminhar', 'fazer cafe']
# refazer = todo ['fazer cafe']
# refazer = todo ['fazer cafe', 'caminhar']
import os
```

```
def listar(tarefas):
    print()
    if not tarefas:
        print('Nenhuma tarefa para listar')
        return

    print('Tarefas:')
    for tarefa in tarefas:
        print(f'\t{tarefa}')
    print()


def desfazer(tarefas, tarefas_refazer):
    print()
    if not tarefas:
        print('Nenhuma tarefa para desfazer')
        return

    tarefa = tarefas.pop()
    print(f'{tarefa=} removida da lista de tarefas.')
    tarefas_refazer.append(tarefa)
    print()
    listar(tarefas)


def refazer(tarefas, tarefas_refazer):
    print()
    if not tarefas_refazer:
        print('Nenhuma tarefa para refazer')
        return

    tarefa = tarefas_refazer.pop()
    print(f'{tarefa=} adicionada na lista de tarefas.')
    tarefas.append(tarefa)
    print()
    listar(tarefas)


def adicionar(tarefa, tarefas):
    print()
    tarefa = tarefa.strip()
    if not tarefa:
        print('Voce nao digitou uma tarefa.')
        return
    print(f'{tarefa=} adicionada na lista de tarefas.')
    tarefas.append(tarefa)
    print()
    listar(tarefas)


tarefas = []
tarefas_refazer = []

while True:
    print('Comandos: listar, desfazer e refazer')
    tarefa = input('Digite uma tarefa ou comando: ')

    comandos = {
        'listar': lambda: listar(tarefas),
        'desfazer': lambda: desfazer(tarefas, tarefas_refazer),
        'refazer': lambda: refazer(tarefas, tarefas_refazer),
        'clear': lambda: os.system('clear'),
        'adicionar': lambda: adicionar(tarefa, tarefas),
    }
    comando = comandos.get(tarefa) if comandos.get(tarefa) is not None else \
        comandos['adicionar']
    comando()

    # if tarefa == 'listar':
    #     listar(tarefas)
    #     continue
    # elif tarefa == 'desfazer':
    #     desfazer(tarefas, tarefas_refazer)
    #     listar(tarefas)
    #     continue
    # elif tarefa == 'refazer':
    #     refazer(tarefas, tarefas_refazer)
    #     listar(tarefas)
    #     continue
    # elif tarefa == 'clear':
    #     os.system('clear')
```

```
    #     continue
    # else:
    #     adicionar(tarefa, tarefas)
    #     listar(tarefas)
    #     continue
```

30 Positional-Only Parameters (/) e Keyword-Only Arguments (*)

```
# *args (ilimitado de argumentos posicionais)
# **kwargs (ilimitado de argumentos nomeados)
#  Positional-only Parameters  Tudo antes da barra deve
# ser APENAS posicional.
# PEP 570  Python Positional Only Parameters
# https://peps.python.org/pep-0570/
#  Keyword-Only Arguments $(*)  *$ sozinho NAO SUGA valores.
# PEP 3102  Keyword Only Arguments
# https://peps.python.org/pep-3102/
def soma(a, b, /, *, c, **kwargs):
    print(kwargs)
    print(a + b + c)


soma(1, 2, c=3, nome='teste')
```