# PYTHON NA PRÁTICA

## APRENDA PYTHON ATRAVÉS DE EXERCÍCIOS

## 2

FERNANDO FELTRIN

# Capa

# PYTHON NA PRÁTICA

## APRENDA PYTHON ATRAVÉS DE EXERCÍCIOS

## 2

FERNANDO FELTRIN

# PYTHON NA PRÁTICA
## Volume 2

Fernando Feltrin

# Avisos

# Sobre o Autor

Fernando Feltrin é Engenheiro da Computação com especializações na área de ciência de dados e inteligência artificial, Professor licenciado para docência de nível técnico e superior, Autor de mais de 10 livros sobre programação de computadores e responsável pelo desenvolvimento e implementação de ferramentas voltadas a modelos de redes neurais artificiais aplicadas à radiologia (diagnóstico por imagem).

# Livros


Disponível em: [Amazon.com.br]

# Sistemática

Este livro é um tratado prático, logo, seu formato é um tanto quanto diferente do convencional por se tratar de um compêndio de exercícios resolvidos e comentados linha a linha de código.

**Crie uma lista com 8 elementos de uma lista de compras de supermercado, por meio de um laço de repetição for liste individualmente cada um dos itens dessa lista.** **<= ENUNCIADO**

```
lista10 = ['Água', 'Açúcar', 'Arroz', 'Pão', 'Leite', 'Massa', 'Carne', 'Refrigerante']

for i in lista10:
  print(i)
```

**<= CÓDIGO**

*Quando estamos a percorrer os elementos de uma lista por meio de um laço for, cada elemento dentro de seu separador será lido individualmente, tendo seu conteúdo (independentemente do tipo de dado) associado a variável temporária i.* **<= EXPLICAÇÃO**

Nesse caso o retorno será:
Água
Açúcar
Arroz
Pão
Leite
Massa
Carne
Refrigerante

**<= RETORNO**

Sendo assim, todos os exercícios estarão padronizados e dispostos no molde acima representado, facilitando assim o entendimento dos mesmos.

## Sumário

5 – Escreva uma estrutura de validação que verifica se o endereço de um site foi digitado corretamente. O endereço escrito deve ser em formato de domínio brasileiro (terminado em .com.br) e para o processo de validação não pode ser usado expressões regulares.

6 – Considerando a lista [11, 22, 33, 44, 55, 66, 77, 88, 99, 'X'], reorganize seus elementos de forma a inverter o primeiro e o último elemento de posição, sem alterar a ordem dos demais elementos.

7 – Reorganize os elementos da lista [-1, 6, -9, -8, 4, 0, -3, 2, 7, 1, 8, -2] em ordem crescente e a partir da mesma, gere duas novas listas, uma contendo apenas os elementos negativos e outra contendo apenas os elementos positivos. Por fim, exiba em tela o conteúdo da lista original e das 3 novas listas geradas.

8 – Supondo que tenhamos uma lista composta por elementos em forma de dicionário, elementos estes aleatórios descrevendo em suas chaves e valores alguns carros com seus respectivos anos de fabricação, reorganize os elementos dessa lista de acordo com um critério pré-estabelecido, nesse caso, o ano de fabricação.

9 – Crie um mecanismo que lê e interpreta uma array, retornando ao usuário o número de elementos da mesma, assim como seu tamanho em bytes e seu endereço de alocação em memória.

10 – Crie um programa que recebe um texto digitado pelo usuário, retornando para o mesmo quantas letras maiúsculas e minúsculas foram identificadas no texto. Apresente dois códigos, um convencional e um reduzido:

11 – Crie um programa que lê um arquivo qualquer, retornando ao usuário os dados referentes a data e hora da última modificação desse arquivo.

12 – Crie um método de classe com parâmetros nomeados, justapostos, com valor padrão, tipo de dado definido na declaração e que retorna um tipo de dado obrigatório e específico no retorno da função, nesta ordem. Como referência, o método

convencional def inventario(item, localizacao, quantidade, preco, categoria) deve ser transformado para um método discriminado, onde cada tipo de parâmetro possui uma identidade e características próprias.

13 – Crie uma classe composta por atributos “somente leitura”, da forma mais reduzida possível:

14 – Crie manualmente um sistema de contagem regressiva, a contar de 10, atualizando seus dados a cada 2 segundos.

15 – Crie uma estrutura de código que simula o sistema de dependência de uma blockchain em um cluster de processamento. Nesse processo é permitido usar de alguma estrutura de repetição, e o cluster inicial deve conter ao menos 4 blocos de origem.

16 – Escreva um programa que, a partir de uma lista de elementos, retorne o tipo de dado de cada elemento. A lista em questão é: lista=[(1,),[1,2,3],(1),{4,5},{'nome':'A','value':2},10,[],1+3j,1.2,True,False,None,'Hello World!'].

17 – Crie uma simples ferramenta de análise exploratória de dados, que recebe uma lista / array, retornando a partir dos dados da mesma os valores referentes ao maior número, menor número, número de elementos totais, soma de todos elementos, média aritmética, mediana, variança e desvio padrão. A lista em questão é: lista = [-1505, 2518, 2333, 4682, -1740, 1067, 995, 22, 2, -1153, -4098, 4560, 2958, 3640, -4154, 2345,4, -1601, 158, -4044, -98, 707, 359, -3088, 527, -3004, -4045, 563, -4137, 4598, -3862, 488, -1, 3445, 3820, 504, -1475, 1626, -1252, 2059, 16, -1472, 2610, -4643, 2912, -2517, -4604, -1476, 3950, -4640, -229, 229, -3452, 4309, 2356, 66, 3728, -1846, -635, -3581, 4457, -2592, -1066, 4,
-1006, -164, 805, -4500, -455, 2245, -4959,2, -2415, 2038, 4512, 1243, 349, -1153, 3623, 631, 2209, -3349, -2417, 4429, -1324, -4101, -1354, 4400, -4968, -4433, 2042, 904, 932, 1331, 4955, -3775, -333, 1012, 2192, -161, -224, 1505, -1615, -2165, 3666, -4253, -358, -3939, -2641, 1228, -608, -2280, 4939, -3355, 1340, -57, 3346,

2686, -1572, 1991, -2351, -3972, -1683, -1509, -2488, 266, -2991, -795, -4032, 2397, 2391, 3654, -1044, -2204, -2440, -1280, 2714, -4151, -1951, 3684, -4251, 3748, -4359, -1436, -2190, 4538, -3563, 1542, 1926, -3940, -2274, -4223, 2504, -4112, 2388]

18 – A partir da lista lista=['a', 'a', 'a', 'b', 'c', 'k', 'a', 1, 2, 3, 4, 'j', 'l', 'm'] gere uma nova lista porém sem elementos repetidos.

19 – Supondo que temos um simples carrinho de compras, em formato de lista composta por alguns elementos onde cada elemento da mesma é um dicionário descrevendo um item, crie um mecanismo que retorna ao usuário o valor total do carrinho, seu item mais caro e também seu item mais barato.

20 – Uma vez apresentada a lista x = [10, 12, 35], apresente ao menos 5 formas de retornar ao usuário a soma dos elementos dessa lista.

21 – Uma prática comum é para certos contextos gerar listas, dicionários, ou qualquer outro tipo de contêiner de dados com alguns valores padrão ou gerados a partir de algum critério. Nessa linha de raciocínio, crie um gerador de dicionário que, com um dado inteiro n fornecido pelo usuário, retorna um dicionário composto por todos números antecessores ao número n e seus valores quando elevados ao quadrado.

22 – Crie um diretório chamado backups, dentro desse diretório, crie uma pasta numerada para cada mês do ano, por exemplo mês_03. Antes de criar a pasta backup, verifique se a mesma já existe no diretório raiz do projeto.

23 – Crie um simples jogo de adivinhação que gera um número aleatório de 0 a 10, em seguida recebe um número do usuário e, caso o mesmo tenha acertado o número, recebe uma mensagem de sucesso, caso contrário, pede que o usuário insira um novo número até acertar a adivinhação.

24 – Crie um template de mala direta contendo uma mensagem, envie a mensagem para 5 pessoas listadas em uma simples base de dados aqui representada pela lista

nomes = ['Anna', 'Paulo', 'Maria', 'Rafael', 'Patricia'] de modo que a mensagem seja personalizada a cada um dos nomes da lista.

25 – Crie um simples mecanismo que lê um determinado texto, nesse caso “Python 3.9”, retornando ao usuário a quantidade de números encontrados no texto e que números são estes, ordenados em ordem posicional (na exata ordem em que foram identificados no texto).

26 – Crie um simples mecanismo validador que recebe do usuário um texto qualquer e retorna ao mesmo se tal texto pertence ou não a uma frase pré-definida, nesse caso: 'Python é uma excelente linguagem de programação!!!'

27 – Escreva uma função que recebe do usuário um número longo qualquer, o convertendo para notação numérica computacional (número original considerado como bytes, convertido de acordo com seu tamanho para kilobytes, megabytes, gigabytes, etc...) e retornando em forma legível ao usuário.

28 – Escreva um programa que recebe do usuário um intervalo de tempo em anos, retornando a partir desta informação o número de ocorrências de um dia específico (nesse caso, as segundas-feiras) dentro de um dia específico do mês (nesse caso, no dia primeiro de cada mês do intervalo de tempo previamente definido).

29 – Crie uma pasta chamada imagens, entre na pasta e retorne seu caminho completo para uma variável. Exiba em tela o caminho completo da pasta imagens.

30 – Crie um mecanismo otimizado de múltipla escolha que recebe do usuário um determinado número em sua forma numérica e retorna para o mesmo o número escrito por extenso:

31 – Crie uma simples função que soma dois números fornecidos pelo usuário, retornando o resultado de tal operação matemática. Para a função soma( _), crie uma documentação acessível pelo usuário diretamente via código, orientando o mesmo a respeito do

funcionamento da função, simulando que tal função é modularizada ou que de alguma forma seu código não é explícito.

32 – Implemente um sistema de carrinho com inserção e remoção de elementos via comando, da forma mais reduzida possível, porém sem fazer uso de compreensão de listas ou funções vazias, apenas estruturas condicionais simples. O sistema deve se manter ativo até que o usuário digite um comando para sair.

33 – Escreva um simples módulo / pacote que inspeciona uma determinada pasta, retornando o nome de todos arquivos de imagem contidos na pasta, caso houver.

34 – Crie um simples mecanismo que valida a inserção de elementos em um dicionário, verificando se o elemento a ser adicionado ao mesmo já existe no dicionário atual.

35 – Uma vez que temos dois conjuntos numéricos iniciais c1 = (2, 4, 6, 8, 10) e c2 = (2, 4, 6, 8, 10) e um terceiro conjunto nomeado como c3, gerado a partir do conteúdo de c2. Equiparando os três conjuntos, verifique se os mesmos possuem mesmo conteúdo e mesmo identificador de memória.

36 – Escreva um programa que procura um certo dado / valor a partir de múltiplos dicionários, conforme dado inserido pelo usuário. Os dicionários em questão devem possuir itens domésticos representados com nome e preço. Retorne ao usuário o item consultado, de qual dicionário foi extraído e seu valor em reais.

37 – Crie três classes representando três funcionários de uma empresa, descreva como atributo de classe primário o cargo, e como atributos secundários seus respectivos nomes e salários, estes dados devem ser imutáveis. Insira os registros destes funcionários em uma lista e exiba em tela o conteúdo da lista de funcionários.

38 – A partir de uma simples base de dados em uma lista, composta por ['1001', '1002', '1003', '1004', '1005', '1006', None, '1008', '1009'], gere uma nova base de dados a partir desta contendo apenas dados

válidos. Retorne ao usuário uma mensagem informando a posição de índice do dado inválido da base de dados original.

39 – Supondo que temos uma lista composta por ['Terça', 'Quarta', 'Quinta', 'Sexta', 'Sabado', 'Domingo', 'Segunda'], reordene os elementos da lista de modo a obter a sequência [‘Segunda’, 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sabado', 'Domingo'].

40 – Uma vez que temos uma lista contendo como elementos próprios uma série de tuplas compostas por uma categoria e por um tipo de exame de imagem, reconstrua essa base de dados em um dicionário, unificando as categorias de exames e agrupando seus subtipos.

41 – Otimize o código anterior fazendo uso de defaultdict( ). Realize a comparação do tempo de processamento necessário para ambos os métodos.

42 – Supondo que temos um dicionário, representando usuários de um sistema, onde cada item do dicionário é um usuário descrito por seu id, nome, sobrenome e e-mail, exiba em tela de forma estruturada (exatamente como escrita no corpo do código) o conteúdo desse dicionário.

43 – Crie um mecanismo que gera um número pseudorrandômico, no intervalo entre 0 e 1, exibindo tal número em tela com 5 casas decimais após a vírgula.

44 – Dada a lista ['python', 'java', 'php', 'c++', 'c#', 'javascript', 'kotlin', 'r'], retorne duas novas listas geradas a partir da lista inicial, com seus elementos reordenados de forma aleatória, porém reproduzível.

45 – Supondo que a lista data = [59.19, 72.05, 66.82, 81.15, 66.33, 94.87, 99.65, 70.13, 55.75, 87.71,
95.43, 93.17, 98.54, 68.31, 59.24, 88.94, 79.44, 83.91, 84.4, 68.21] é o retorno dos últimos 20 registros de um sensor qualquer, retorne a média simples, a média harmônica e a média geométrica destes valores. Os resultados devem conter a precisão de 5 casas decimais.

46 – Supondo que para um determinado propósito temos uma classe de nome Menu, que quando instanciada recebe um dado / valor que deve obrigatoriamente ser de tipo numérico para criação de seu respectivo objeto / atributo de classe. A validação por sua vez deve ocorrer a partir do método construtor da classe.

47 – Como bem sabemos, operações matemáticas em Python realizadas a partir de valores numéricos muito próximos tendem a gerar números com casas decimais, valores em casas decimais normalmente “arredondados” por questões de performance. Porém, em computação científica pode ser necessário números com grande número de casas decimais pois estes valores, quanto mais extensos, simbolizam maior precisão. Realize a simples soma de dois valores 0.1 e 0.2, retornando um terceiro número de no mínimo 20 casas decimais.

48 – Uma vez que temos as frações 1/4 e 5/8, realize a soma dessas frações, exibindo em tela detalhadamente seus numeradores, denominadores e o próprio resultado da soma.

49 – Crie um mecanismo de desempacotamento de dados, capaz de reduzir listas aninhadas para uma só lista comum sem alterar o comportamento de seus elementos. A lista a ser usada de referência é: [[1, 2, [3, 4, 5, 6, 7, 8]], [9, 10, 11, 12]].

50 – Implemente dois mecanismos de extração de dados a partir de dicionários aninhados, sendo ao menos um deles o método convencional, utilizando da notação padrão de extração de dados a partir das chaves de um dicionário.

Considerações Finais

# Introdução

Se tratando de programação, independentemente da linguagem, um dos maiores erros dos estudantes ou entusiastas da área é o fato de os mesmos apenas se limitarem a replicar os exemplos dos conceitos da linguagem, não praticando de fato, seja explorando outras possibilidades de mesmo propósito, seja não resolvendo exercícios que estimulem a aplicação da lógica e implementação das ferramentas que a linguagem de programação oferece para resolução de problemas computacionais.

Sendo assim, o intuito deste pequeno livro é trazer à tona desde os conceitos mais básicos da linguagem Python até tópicos específicos e avançados por meio de exercícios, de modo que cada conceito será revisado conforme a necessidade de sua aplicação, assim como para programadores que já possuem uma bagagem de conhecimento, tais exemplos servirão para os mesmos realizar engenharia reversa nos conceitos entendendo os porquês de cada estrutura de código.

Esse livro foi escrito de forma que os exercícios estão dispostos de forma gradual no que diz respeito à sua dificuldade, todo e qualquer exercício terá suas devidas explicações para cada linha de código, porém de forma gradual à medida que avançaremos pelos exercícios iremos cada vez nos atendo mais às suas particularidades do que conceitos iniciais. Em função disso, faça os exercícios em sua ordem, não avançando quando encontrada alguma dificuldade, assim como quando se sentir confiável, tente outros métodos de resolução dos mesmos exercícios.

Dessa forma, garanto que você aprenderá de uma vez por todas como aplicar seus conhecimentos em problemas computacionais reais.

Então... vamos direto para prática!

# Exercícios

**1 – Na frase “*A Radiologia é a ciência que estuda a anatomia por meio de exames de imagem*.” realize a contagem dos caracteres de sua composição, retornando quais os 6 caracteres mais recorrentes, e quantas vezes os mesmos aparecem na frase acima:**

```
from collections import Counter

texto = 'A Radiologia é a ciência que estuda a anatomia por meio de exames d
e imagem'

a1, a2, a3, a4, a5, a6 = Counter(texto).most_common(6)

print(f'A letra mais recorrente é: "{a2[0]}", repetida {a2[1]} vezes')
print(f'A letra mais recorrente é: "{a3[0]}", repetida {a3[1]} vezes')
print(f'A letra mais recorrente é: "{a4[0]}", repetida {a4[1]} vezes')
print(f'A letra mais recorrente é: "{a5[0]}", repetida {a5[1]} vezes')
print(f'A letra mais recorrente é: "{a6[0]}", repetida {a6[1]} vezes')
```

*Sempre que estamos tratando de um texto / string, sabemos que cada caractere do mesmo é um elemento indexado assim como em uma lista, diferenciando apenas a sintaxe em ambos tipos de dados. Sendo assim, é comum raciocinarmos que a melhor forma de se extrair informações de tais elementos seja aplicando um laço de repetição for percorrendo cada um dos mesmos a partir de seus valores de índice. Porém, se tratando especificamente da realização de contagem do número de elementos, podemos simplificar / automatizar esse processo combinando os métodos Counter( ) e most_common( ) da biblioteca collections.*

*Sendo assim, inicialmente importamos o método Counter da biblioteca collections através de from e import.*

*Em seguida declaramos uma variável de nome texto que por sua vez recebe como atributo a frase do enunciado da questão em forma de string.*

*Na sequência criamos 6 novas variáveis, de nomes a1, a2, a3, a4, a5 e a6, respectivamente, que instanciam e inicializam o método Counter( ) por sua vez parametrizado com nossa variável texto, aplicando sobre essa função inicial o método most_common( ), aqui parametrizado com o valor 6 para que sejam mapeados os 6 elementos mais recorrentes da string em questão. Dessa forma, estamos indiretamente desempacotando os elementos de texto, mapeando-os conforme seu número de ocorrências, por fim separando e atribuindo cada grupo de elementos para suas respectivas variáveis, sendo a1 recebendo o elemento mais recorrente seguido de seu número de recorrência até a6 que recebe como atributo o elemento menos recorrente com seu respectivo valor de recorrência.*

*Terminada esta etapa, podemos exibir em tela ao usuário através de nossa função print( ), por sua vez parametrizada com uma f'string onde em suas máscaras de substituição instanciamos as devidas variáveis e sua posição de índice, sendo posição 0 retornando o caractere em si e a posição 1 o número de recorrência.*

Nesse caso, o retorno será:

**A letra mais recorrente é: "a", repetida 11 vezes**
**A letra mais recorrente é: "e", repetida 8 vezes**
**A letra mais recorrente é: "i", repetida 7 vezes**
**A letra mais recorrente é: "o", repetida 5 vezes**
**A letra mais recorrente é: "m", repetida 5 vezes**

---

**2 – Crie uma estrutura de código que lê o conteúdo interno de uma função, retornando o número de identificação da função e o número de variáveis que a mesma possui alocadas em memória.**

```
def minha_funcao():
    x = 1
    y = 2
    nome = 'Tania'
    lista = [1, 2, 3]
    return f'{x}, {y}, {nome}, {lista}'

print(minha_funcao())

var_locais = minha_funcao.__code__.co_nlocals

print(f'A função {minha_funcao} possui {var_locais} variáveis')
```

*Para certos contextos, pode ser relevante extrair o endereço de alocação de memória de um determinado objeto (lembrando que em Python tudo é objeto, inclusive funções), assim como o número de variáveis locais que tal objeto possui instância, nesse caso, uma função.*

*Para isso, podemos instanciar o objeto em questão, aplicando sobre o mesmo o método interno co_nlocals para retornar o número de objetos internos assim como podemos ler __code__ para assim ver o endereço o qual referencia tal objeto e todo seu escopo.*

*Partindo para o código, criada uma função de nome minha_funcao( ), sem parâmetros mesmo, em seu corpo declaramos 4 variáveis quaisquer, de tipos de dados atribuídos diferentes apenas para fins de exemplo, retornando o conteúdo de tais variáveis toda vez que a função é inicializada.*

*Logo após, repassando como parâmetro para print( ) a função minha_funcao( ), será exibido em tela os últimos dados / valores atribuídos para suas respectivas variáveis locais.*

*Na sequência declaramos uma nova variável, dessa vez de nome var_locais, que recebe como atributo o conteúdo de minha_funcao quando aplicado sobre a mesma __code__ e co_nlocals.*

*Por fim, exibindo em tela o conteúdo de var_locais através da função print( ), é esperado que sejam retornados os dados pedidos pelo enunciado dessa questão.*

Não havendo nenhum erro de sintaxe, o retorno será:

**1, 2, Tania, [1, 2, 3]**
**A função <function minha_funcao at 0x7f6c69b1c560> possui 4 variáveis**

---

**3 – Uma vez que temos a lista [“maçã”, “banana”, “laranja”, “abacate”, “pêssego”], busque extrair o segundo e o quarto elemento da lista de forma ainda mais reduzida e otimizada do que quando utilizamos list comprehension.**

```
frutas = ["maçã", "banana", "laranja", "abacate", "pêssego"]

from operator import itemgetter

n_citricas = list(itemgetter(1, 3)(frutas))

print(n_citricas)
```

*Aqui temos um cenário interessante, pois de acordo com o enunciado da questão, devemos buscar uma alternativa ainda mais reduzida e ainda mais otimizada do que list comprehension, sendo que “compreensão” aplicada a listas (e outros contêineres de dados) costuma ser o método “reduzido e otimizado”.*

*Porém, como bem sabemos, Python oferece um gigantesco leque de possibilidades no que diz respeito a forma como realizamos ações em nossos códigos, de modo que através de duas bibliotecas, módulos e pacotes (muitos inclusive produzidos por terceiros) temos à disposição ferramentas de fato aprimoradas quando comparadas as ferramentas padrão da linguagem.*

*Sendo assim, cumprindo o desafio proposto pela questão, uma forma simples de chegarmos ao resultado esperado é realizando o desempacotamento / extração dos elementos em questão da lista por meio do método itemgetter( ) do módulo operator.*

*Direto ao código, inicialmente declaramos uma variável de nome frutas que recebe a lista descrita no enunciado da questão como seu atributo.*

*Em seguida importamos itemgetter do módulo operator conforme o código escrito na linha 3.*

*Na sequência criamos uma nova variável, dessa vez nomeada como n_citricas, que por sua vez recebe em forma de lista o retorno produzido pelo método itemgetter( ), por sua vez parametrizado com os números de índice seguido da instância encadeada da variável frutas.*

*Por fim, exibimos em tela o conteúdo da variável n_citricas através de nossa função print( ), nesse caso retornando:*

**['banana', 'abacate']**

---

**4 – Crie uma estrutura de código de validação de uma senha digitada pelo usuário. Após inserida a senha, a mesma deve ser inspecionada em busca das seguintes características: Conter entre 6 e 16 caracteres, conter ao menos uma letra minúscula, conter ao menos um número, conter ao menos uma letra maiúscula e conter ao menos um caractere especial ($#@*!&).**

```
import re

senha = input('Insira sua nova senha: ')
x = True

while x:
    if (len(senha) < 6 or len(senha) > 16):
        print('A senha deve conter entre 6 e 16 caracteres')
        break
    elif not re.search("[a-z]", senha):
        print('A senha deve ter ao menos uma letra minúscula')
        break
    elif not re.search("[0-9]", senha):
        print('A senha deve conter ao menos um número')
        break
    elif not re.search("[A-Z]", senha):
        print('A senha deve conter ao menos uma letra maiúscula')
        break
    elif not re.search("[$#@*!&]", senha):
        print('A senha deve conter ao menos um caractere especial')
        break
    elif re.search("\s", senha):
        break
    else:
        print('Senha cadastrada com sucesso!!!')
        x = False
        break
```

```
if x:
    print('Senha inválida!!!')
```

*Sempre que temos algum problema proposto na interpretação de elementos que compõem uma string, realizando algum tipo de validação, podemos simplificar tal processo conciliando estruturas condicionais e expressões regulares. Existem logicamente diferentes formas de se solucionar o mesmo problema, porém de acordo com os tipos de validação exigidos pela questão é interessante (ao menos para fins didáticos) realizarmos uma validação para cada tópico proposto.*

*Logo no início de nosso código realizamos a importação da biblioteca re, referente as funções específicas aplicáveis a r'strings (expressões regulares).*

*Dando sequência declaramos uma variável de nome senha, que através do método input( ) pede ao usuário que o mesmo digite uma senha.*

*É criada também uma variável de nome x, que inicialmente apenas recebe como atributo True, pois essa será uma variável de controle para estrutura condicional que criaremos em seguida.*

*Uma vez que estaremos a realizar diversas validações sobre os dados atribuídos a variável senha, é interessante fazer uso de uma variável de controle associada a estrutura while, pois assim teremos condições de testar várias validações, apenas permitindo que o código dê sequência em alguma ação quando todas exigências impostas forem cumpridas para os dados da variável em questão.*

*Sendo assim, é criada uma estrutura condicional while x, que implicitamente está criando uma primeira camada de estrutura condicional, validando inicialmente que enquanto o valor de x for*

*True, os blocos aninhados a esta estrutura condicional deverão ser executados.*

*Em seguida, criamos uma estrutura condicional if, que através de uma estrutura condicional composta (de 2 ou mais proposições) verifica se o tamanho de senha é menor que 6 ou maior que 16 através do método len( ). Repare que caso uma ou outra proposição seja falsa, é exibida uma mensagem ao usuário e acionado o gatilho break, para que já neste ponto o processo da estrutura condicional seja interrompido.*

*Também é criada uma estrutura condicional elif, que agora realiza a negação lógica verificando se o retorno do método re.search( ), por sua vez parametrizado em justaposição com os elementos a serem buscados (nesse caso, caracteres minúsculos de a até z) e com a instância da variável a ser inspecionada. Caso esta proposição seja verdadeira, confirmando que no conteúdo atribuído a variável senha não existe nenhuma letra minúscula de a até z, a estrutura while é interrompida.*

*Nos mesmos moldes são criadas outras estruturas de validação, usando de negação lógica e do método re.search( ) para buscar se na senha gerada pelo usuário não existem números de 0 a 9, letras maiúsculas de A até Z, os caracteres especiais conforme enunciado da questão e por fim se o texto digitado contém espaços em branco via "\s".*

*Na sequência é criado o else, onde caso todas as proposições anteriores das estruturas condicionais tenham sido validadas conforme esperado, é exibido em tela ao usuário uma mensagem de sucesso, atualizando o valor de nossa variável x para False e encerrando a cadeia de validações.*

*Dessa forma, de modo simples e objetivo realizamos todas as devidas validações de acordo com as exigências levantadas pela questão.*

Apenas como exemplo, digitando 123 o retorno será:

**Insira sua nova senha: 123**

**A senha deve conter entre 6 e 16 caracteres**
**Senha inválida!!!**

Digitando 123abc o retorno será:

**Insira sua nova senha: 123abc**
**A senha deve conter ao menos uma letra maiúscula**
**Senha inválida!!!**

Digitando 1A2b3c o retorno será:

**Insira sua nova senha: 1A2b3c**
**A senha deve conter ao menos um caractere especial**
**Senha inválida!!!**

Por fim, digitando 1A2b3# o retorno será:

**Insira sua nova senha: 1A2b3#**
**Senha cadastrada com sucesso!!!**

---

**5 – Escreva uma estrutura de validação que verifica se o endereço de um site foi digitado corretamente. O endereço escrito deve ser em formato de domínio brasileiro (terminado em .com.br) e para o processo de validação não pode ser usado expressões regulares.**

```
endereco = input('Digite o endereço completo de seu site:')

while ((endereco.startswith('www.') and endereco.endswith('.com.br'))) != True:
    print('O endereço deve incluir "www" e ".com.br"')
    endereco = input('Digite o endereço do site: ')

print(f'{endereco} é um endereço válido.')
```

*Ao se tratar de validar o endereço de um site, especificamente se fazer uso de expressões regulares, estamos tratando o prefixo e o sufixo de um texto, nesse caso inserido pelo usuário.*

*Nessa linha de raciocínio, uma forma simples de resolver este problema é validando um prefixo e um sufixo específico através dos métodos startswith( ) e endswith( ), que verificam justamente com que string um texto inicia e termina, respectivamente.*

*Partindo para o código, inicialmente é declarada uma variável de nome endereco, que recebe como atributo algo digitado pelo usuário através do método input( ).*

*Na sequência é criada uma estrutura condicional composta associada ao laço de repetição while, verificando se a string atribuída a variável endereco é iniciada em "www." e terminada em ".com.br". Enquanto essas duas proposições forem diferentes de True, é exibida em tela uma mensagem de instrução ao usuário via print( ) e a variável endereco é instanciada novamente, mais uma vez pedindo que o usuário digite o endereço do site.*

*Quando as duas proposições forem validadas como verdadeiras, a estrutura de repetição é interrompida.*

Nesse caso, digitando google.com o retorno será:

**O endereço deve incluir "www" e ".com.br"**
**Digite o endereço do site:**

Digitando www.google.com.br o retorno será:

**Digite o endereço do site: www.google.com.br**
**www.google.com.br é um endereço válido.**

---

**6 – Considerando a lista [11, 22, 33, 44, 55, 66, 77, 88, 99, 'X'], reorganize seus elementos de forma a inverter o primeiro e o último elemento de posição, sem alterar a ordem dos demais elementos.**

```
lista1 = [11, 22, 33, 44, 55, 66, 77, 88, 99, 'X']

print(f'Lista original: {lista1}')

temp = lista1[0]
lista1[0] = lista1[-1]
lista1[-1] = temp

print(f'Lista final: {lista1}')
```

*Para rearranjar elementos de uma lista podemos simplesmente trabalhar manipulando os mesmos através de seu número de índice. Lembrando que o índice de listas em Python se inicia em 0 e que podemos fazer referência direta ao último elemento de uma lista utilizando -1.*

*Partindo para resolução do problema, logo no início de nosso código declaramos uma variável de nome lista1, que recebe como atributo a lista descrita no enunciado da questão, respeitando a sintaxe de lista.*

*Na sequência criamos uma nova variável, dessa vez nomeada como temp, que recebe como atributo apenas o primeiro elemento / elemento situado na posição 0 de índice da variável lista1.*

*Em seguida instanciamos lista1, especificamente sua posição de índice 0, atualizando seu dado / valor com o dado / valor situado na última posição de índice de lista1 através da notação [-1].*

*Por fim, instanciamos lista1, especificamente o último elemento da mesma, atualizando seu valor com o dado / valor que havíamos*

*guardado anteriormente na variável temp. Dessa forma, simplesmente trocamos o primeiro e o último elementos da lista sem a necessidade de iterar sobre os demais elementos da mesma.*

*Encerrando, exibimos em tela novamente através da função print( ) o conteúdo de lista1 após feitas as devidas alterações.*

Não havendo nenhum erro de sintaxe, o retorno será:

**Lista original: [11, 22, 33, 44, 55, 66, 77, 88, 99, 'X']**
**Lista final: ['X', 22, 33, 44, 55, 66, 77, 88, 99, 11]**

---

**7 – Reorganize os elementos da lista [-1, 6, -9, -8, 4, 0, -3, 2, 7, 1, 8, -2] em ordem crescente e a partir da mesma, gere duas novas listas, uma contendo apenas os elementos negativos e outra contendo apenas os elementos positivos. Por fim, exiba em tela o conteúdo da lista original e das 3 novas listas geradas.**

```
lista1 = [-1, 6, -9, -8, 4, 0, -3, 2, 7, 1, 8, -2]

print(f'Lista original: {lista1}')

lista2 = [x for x in lista1 if x < 0] + [x for x in lista1 if x >= 0]
lista3 = [x for x in lista1 if x < 0]
lista4 = [x for x in lista1 if x >= 0]

print(f'Lista rearranjada: {lista2}')
print(f'Elementos negativos: {lista3}')
print(f'Elementos positivos: {lista4}')
```

*Este é um problema de simples resolução, onde podemos percorrer os elementos da lista original através de um laço for, mapeando cada um dos elementos, por fim classificando os mesmos de acordo com os critérios exigidos pela questão.*

*Para isso, inicialmente criamos uma variável de nome lista1, que recebe como seu atributo a lista proposta pela questão. Em seguida exibimos seu conteúdo original em tela simplesmente a repassando como atributo para a função print( ).*

*Na sequência criamos uma nova variável de nome lista2, que como atributo recebe a concatenação de dois laços for inseridos em forma de list comprehension, sendo o primeiro laço mapeando a cada ciclo de repetição cada um dos elementos de valor menor que 0, da mesma forma o segundo laço mapeando e extraindo cada um*

*dos elementos de valor igual ou maior que 0. Dessa forma, geramos uma lista atribuída para a variável lista2 contendo todos os elementos de lista1, organizados de forma crescente.*

*Em seguida criamos duas novas variáveis de nomes lista3 e lista4, respectivamente, que reaproveitam exatamente os mesmos códigos criados em list comprehension para os mapeamentos anteriores, extraindo para si os elementos menores e maiores que 0, representando os elementos negativos e positivos conforme pedido através do enunciado da questão.*

*Por fim, através da função print( ) exibimos os devidos conteúdos das variáveis lista2, lista3 e lista4.*

Nesse caso o retorno será:

**Lista original: [-1, 6, -9, -8, 4, 0, -3, 2, 7, 1, 8, -2]**
**Lista rearranjada: [-1, -9, -8, -3, -2, 6, 4, 0, 2, 7, 1, 8]**
**Elementos negativos: [-1, -9, -8, -3, -2]**
**Elementos positivos: [6, 4, 0, 2, 7, 1, 8]**

---

**8 – Supondo que tenhamos uma lista composta por elementos em forma de dicionário, elementos estes aleatórios descrevendo em suas chaves e valores alguns carros com seus respectivos anos de fabricação, reorganize os elementos dessa lista de acordo com um critério pré-estabelecido, nesse caso, o ano de fabricação.**

```
def f(crit):
  return crit['ano']

carros = [
  {'carro': 'Celta', 'ano': 2005},
  {'carro': 'Gol', 'ano': 2000},
  {'carro': 'Palio', 'ano': 2019},
  {'carro': 'Corsa', 'ano': 2011},
  {'carro': 'Fiesta', 'ano': 2009}]

carros.sort(key=f)

print(carros)
```

*Quando estamos estudando coleções / contêineres de dados, especificamente listas e dicionários, de início pode nos parecer um tanto quanto abstrato quando temos cenários com listas dentro de dicionários e vice-versa.*

*Porém, se tratando da organização de elementos em camadas de dicionários dentro de listas, ou como elementos de uma lista, o processo pode ser simplificado através do método sort( ), que receberá para seu parâmetro nomeado key algum critério a ser respeitado para a organização.*

*Diretamente na prática, logo de início criamos uma simples função de nome f( ), que obrigatoriamente receberá um critério a ser*

*repassado pelo usuário. Dentro do corpo dessa função simplesmente retornamos esse critério, aqui já descrito como por padrão 'ano'.*

*Na sequência criamos uma variável de nome carros, atribuindo para a mesma uma lista composta de elementos em forma de chaves e valores de um dicionário conforme é pedido na questão. Em tais elementos, alimentamos suas chaves com alguns modelos de carros e seus respectivos anos de fabricação.*

*Por fim, para reorganizar os elementos quanto a seu ano de fabricação, instanciamos a variável carros, aplicando diretamente sobre a mesma o método sort( ), por sua vez parametrizando key com nossa função f (aqui repassada sem parênteses para que assim apenas usemos da função quando necessário, e não imediatamente quando instanciada).*

*Por fim, exibimos em tela via print( ) o conteúdo da variável carros.*

**[{'carro': 'Gol', 'ano': 2000}, {'carro': 'Celta', 'ano': 2005}, {'carro': 'Fiesta', 'ano': 2009}, {'carro': 'Corsa', 'ano': 2011}, {'carro': 'Palio', 'ano': 2019}]**

---

**9 – Crie um mecanismo que lê e interpreta uma array, retornando ao usuário o número de elementos da mesma, assim como seu tamanho em bytes e seu endereço de alocação em memória.**

```
from array import *

lista1 = array('i', [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21])

print(f'lista1 é composta de {len(lista1)} elementos')

print(f'lista1 possui um tamanho de {lista1.itemsize} bytes')

print(f'lista1 está alocada na memória no endereço: {lista1.buffer_info()[0]}')
```

*Existe uma vasta gama de bibliotecas, módulos e pacotes para manipulação de arrays em Python, porém, para fins de exemplo, vamos usar do módulo nativo array para através de seus métodos produzir os dados solicitados pela questão.*

*Sendo assim, na primeira linha de nosso código importamos tudo do módulo array através do comando from array import *.*

*Logo após declaramos uma variável de nome lista1, que por sua vez instancia e inicializa o método array( ) para criação de uma array padrão, nesse caso, composta por elementos todos de tipo int, e da lista de elementos em si.*

*Para resolver o primeiro problema da questão, podemos diretamente como parâmetro para a função print( ), fazendo uso de uma f'string, inserir em sua máscara de substituição o método len( ), por sua vez parametrizado com o conteúdo de lista1 para assim exibir em tela o número de elementos da array.*

*Da mesma forma, inserindo na máscara de substituição de uma nova f'string de uma nova função print( ) a variável lista1,*

*aplicando sobre a mesma o método itemsize, será retornado o tamanho em bytes deste objeto em questão.*

*Por fim, nos mesmos moldes fazendo referência a variável lista1, aplicando sobre a mesma o método buffer_info( ), será gerado como retorno padrão o endereço de alocação do objeto em questão em memória.*

Não havendo nenhum erro de sintaxe, o retorno será:

**lista1 é composta de 11 elementos**
**lista1 possui um tamanho de 4 bytes**
**lista1 está alocada na memória no endereço: 139907527601968**

**----------**

**10 – Crie um programa que recebe um texto digitado pelo usuário, retornando para o mesmo quantas letras maiúsculas e minúsculas foram identificadas no texto. Apresente dois códigos, um convencional e um reduzido:**

Método convencional:

```
texto = input('Digite um texto: ')

maiusculas = []
minusculas = []

for letra in texto:
    if (letra.islower()):
        minusculas.append(letra)
    elif (letra.isupper()):
        maiusculas.append(letra)

print(f'No texto {texto} foram encontradas as seguintes letras maiúsculas {maiu
sculas}')
print(f'No texto {texto} foram encontradas {len(maiusculas)} letras maiúsculas')

print(f'No texto {texto} foram encontradas as seguintes letras minúsculas {minu
sculas}')
print(f'No texto {texto} foram encontradas {len(minusculas)} letras minusculas')
```

*Vamos ao código, onde logo na primeira linha declaramos uma variável de nome texto, que por sua vez através do método input( ), recebe algo digitado pelo usuário.*

*Em seguida, criamos duas novas variáveis de nomes maiusculas e minusculas, respectivamente, que inicialmente recebem como seus atributos uma lista vazia, a ser atualizada à medida que caracteres do texto digitado pelo usuário são mapeados.*

*Na sequência criamos um laço de repetição for, que percorrerá cada elemento de texto, e a cada ciclo de repetição, submete o elemento mapeado a uma estrutura condicional onde, caso o mesmo seja identificado e validado como True pelo método islower( ), será inserido na lista minusculas via método append( ).*

*Caso a primeira proposição não seja verdadeira, o elemento é novamente testado, agora submetido a validação como caractere maiúsculo pelo método isupper( ). Caso essa proposição seja verdadeira, o mesmo é inserido na lista maiusculas através do método append( ).*

*Por fim, diretamente como parâmetro para função print( ) geramos alguns retornos, exibindo em tela ao usuário os caracteres mapeados e o número de letras maiúsculas e minúsculas.*

Nesse caso, digitando Fernando Belome Feltrin, o retorno será:

**Digite um texto: Fernando Belome Feltrin**
**No texto Fernando Belome Feltrin foram encontradas as seguintes letras maiúsculas ['F', 'B', 'F']**
**No texto Fernando Belome Feltrin foram encontradas 3 letras maiúsculas**
**No texto Fernando Belome Feltrin foram encontradas as seguintes letras minúsculas ['e', 'r', 'n', 'a', 'n', 'd', 'o', 'e', 'l', 'o', 'm', 'e', 'e', 'l', 't', 'r', 'i', 'n']**
**No texto Fernando Belome Feltrin foram encontradas 18 letras minusculas**

Método reduzido:

```
texto = input('Digite um texto: ')

minusculas = [letra for letra in texto if letra.islower()]
maiusculas = [letra for letra in texto if letra.isupper()]

print(f'No texto {texto} foram encontradas as seguintes letras maiúsculas {maiu
sculas}')
print(f'No texto {texto} foram encontradas {len(maiusculas)} letras maiúsculas')
```

```
print(f'No texto {texto} foram encontradas as seguintes letras minúsculas {minu
sculas}')
print(f'No texto {texto} foram encontradas {len(minusculas)} letras minusculas')
```

*Para reduzir e otimizar o código anterior, não somente no quesito tamanho, mas também sua performance, uma vez que estamos trabalhando sobre dados em formato de contêiner de dados indexados, podemos usar de list comprehension sobre tais dados para assim aplicar o mapeamento e a validação de cada caractere lido sobre tais dados.*

*Para isso, mantemos a primeira estrutura criada no código anterior, referente a interação com o usuário e entrada de dados via input( ) para nossa variável texto.*

*Em seguida, para a variável minusculas é atribuído o retorno gerado pela expressão letra for letra in texto if letra.islower( ), em outras palavras, aqui é aplicado um laço de repetição for que percorrerá cada letra de texto, extraindo apenas os que forem validados como true pelo método islower( ).*

*Exatamente o mesmo processo é realizado para variável maiusculas, alterando também o método aplicado para isupper( ).*

*Dessa forma, reduzimos 8 linhas de código em 2 com as exatas mesmas funcionalidades.*

*Por fim, nos mesmos moldes como feito no código anterior, através de funções print( ) exibimos os devidos caracteres e seu número de ocorrências no texto digitado pelo usuário.*

Não havendo nenhum erro de sintaxe, o retorno será:

**Digite um texto: Fernando Belome Feltrin**

**No texto Fernando Belome Feltrin foram encontradas as seguintes letras maiúsculas ['F', 'B', 'F']**
**No texto Fernando Belome Feltrin foram encontradas 3 letras maiúsculas**

**No texto Fernando Belome Feltrin foram encontradas as seguintes letras minúsculas ['e', 'r', 'n', 'a', 'n', 'd', 'o', 'e', 'l', 'o', 'm', 'e', 'e', 'l', 't', 'r', 'i', 'n']**
**No texto Fernando Belome Feltrin foram encontradas 18 letras minusculas**

---

**11 – Crie um programa que lê um arquivo qualquer, retornando ao usuário os dados referentes a data e hora da última modificação desse arquivo.**

```
import os
import time

def ultima_modificacao(arquivo):
    status = os.stat(arquivo)
    data = time.localtime((status.st_mtime))
    ano = data[0]
    mes = data[1]
    dia = data[2]
    hora = data[3]
    minuto = data[4]
    segundo = data[5]

    str_ano = str(ano)[0:]

    if (mes <= 9):
        str_mes = '0' + str(mes)
    else:
        str_mes = str(mes)

    if (dia <= 9):
        str_dia = '0' + str(dia)
    else:
        str_dia = str(dia)

    return (str_dia + '-' + str_mes + '-
' + str_ano + ' / ' + str(hora) + ':' + str(minuto) + ':' + str(segundo))

print('Ultima modificação realizada em: ', ultima_modificacao('testes.txt'))
```

*Uma vez que estamos a manipular datas, horas ou qualquer outra informação temporal, podemos perfeitamente fazer uso de algumas ferramentas pré-moldadas da biblioteca nativa time, de modo a interagir com o sistema, com estruturas de nosso código e até mesmo com arquivos carregados para obter dados.*

*Sendo assim, vamos buscar extrair as características exigidas pelo enunciado da questão da forma mais simples e direta possível.*

*Para tal feito, inicialmente importamos as bibliotecas, módulos e pacotes os quais faremos uso ao longo de nosso código, nesse caso, importando os e time.*

*Logo após, é criada uma função chamada ultima_modificacao( ), que receberá obrigatoriamente um arquivo a ser interpretado.*

*Dentro do corpo dessa função, vamos criar uma série de variáveis as quais guardarão dados únicos extraídos através dos métodos os.stat( ) e time.localtime( ). Primeira variável nomeamos como status, que como atributo instancia e inicializa o método os.stats( ) por sua vez parametrizado com o arquivo em questão. Nesta etapa estamos a fazer uma leitura de todos status mapeados no arquivo a nível do próprio sistema operacional.*

*De modo parecido, criamos uma nova variável, agora de nome data, que recebe como seu atributo o retorno do método time.localtime( ), aqui parametrizado com status.st_mtime para que seja feita a leitura do horário real do sistema no momento da manipulação do arquivo. O retorno padrão deste método será uma série de informações as quais iremos desmembrar e compreender logo em seguida em suas respectivas variáveis.*

*Dando continuidade, é criada uma variável de nome ano, que recebe como atributo o último dado / valor atribuído a data na posição 0 de seu índice. Também são criadas as variáveis ano, mes, dia, hora, minuto e segundo, que recebem dados únicos a partir das posições de índice 1, 2, 3, 4 e 5 da variável data, respectivamente.*

*Em seguida é necessário realizar algumas simples validações para que os valores sejam exibidos corretamente, haja visto que os*

*dados iniciais estarão em formato americano, e além disso, para deixarmos todos os valores equivalentes no que diz respeito ao número de caracteres é necessário a inserção de alguns caracteres 0 em posições específicas. Ao final deste processo teremos valores para ano, mês e dia dentro da formatação a qual precisamos.*

*Para esse processo, criamos uma variável de nome str_ano, que basicamente recebe os dados / valores de ano em forma de string de acordo com o "fatiamento" aplicado.*

*Na sequência criamos uma estrutura condicional que verifica se o valor atribuído para variável mes é igual ou menor a 9, caso seja, é criada uma variável de nome str_mes que recebe a concatenação da string 0 com o número do mês também convertido para string. Raciocine que nesta etapa, o que estamos fazendo é simplesmente pegar o valor de mês (por exemplo 4) e o reescrever como "04".*

*Exatamente o mesmo processo é replicado para a variável dia, gerando a partir da mesma uma nova variável de nome str_dia com dado / valor atribuído formatado para sempre conter 2 caracteres em sua composição.*

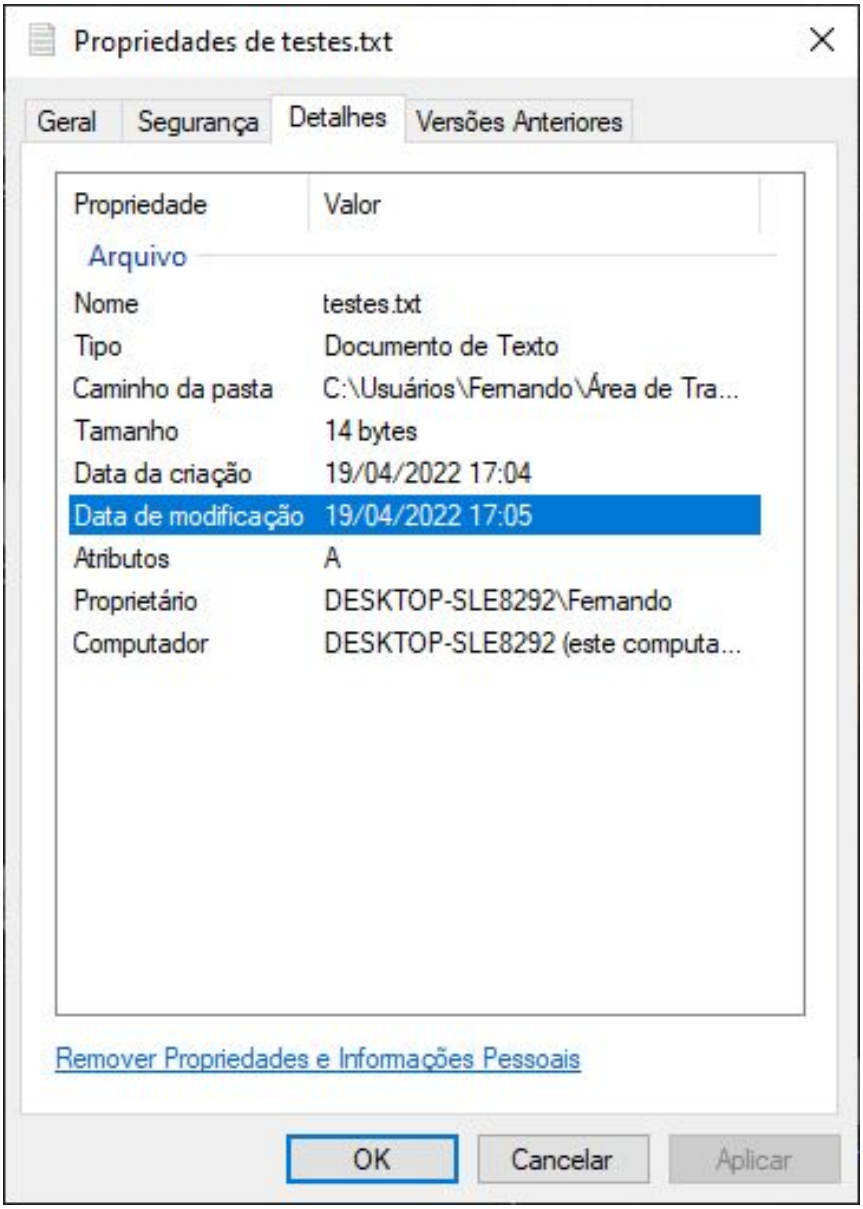
*Por fim, geramos como retorno da função ultima_modificacao( ) uma string composta pela concatenação dos dados de str_dia, str_mes, str_ano, hora, minuto e segundo (nesta ordem específica para ficar em formato de data / hora brasileira).*

*Finalizando, de volta ao escopo geral do código através de uma função print( ) exibimos em tela os dados os quais buscamos a partir do retorno da função ultima_modificacao( ), aqui parametrizada com o arquivo em questão, nesse caso, um arquivo chamado testes.txt situado no mesmo diretório do código.*

Executando o código, o retorno será:

**Ultima modificação realizada em:  19-04-2022 / 17:5:6**

Apenas para fins de comparação:


Propriedades de testes.txt
Geral
Segurança
Detalhes
Versões Anteriores
Propriedade
Valor
Arquivo
Nome
testes.txt
Tipo
Documento de Texto
Caminho da pasta
C:\Usuários\Fernando\Área de Tra...
Tamanho
14 bytes
Data da criação
19/04/2022 17:04
Data de modificação
19/04/2022 17:05
Atributos
A
Proprietário
DESKTOP-SLE8292\Fernando
Computador
DESKTOP-SLE8292 (este computa...
Remover Propriedades e Informações Pessoais
OK
Cancelar
Aplicar

---

**12 – Crie um método de classe com parâmetros nomeados, justapostos, com valor padrão, tipo de dado definido na declaração e que retorna um tipo de dado obrigatório e específico no retorno da função, nesta ordem. Como referência, o método convencional def inventario(item, localizacao, quantidade, preco, categoria) deve ser transformado para um método discriminado, onde cada tipo de parâmetro possui uma identidade e características próprias.**

```
def inventario(item, localizacao, quantidade, preco, categoria):
  pass

# Método discriminado
def inventario(item,
               localizacao = 'Estoque',
               quantidade = 0,
               preco: float = 0.01,
               categoria: list['Perifericos',
                               'Acessorios',
                               'Outros'],
               **kwargs) -> np.ndarray:
  pass
```

*Pois bem, como bem sabemos, em Python tudo é muito dinâmico, a ponto de o tipo de dado e o comportamento do mesmo poder ser alterado a qualquer momento, em qualquer escopo, sem grandes problemas de compatibilidade. O que o exercício propõe é justamente explorarmos as possibilidades no que diz respeito a manualmente configurar o tipo de parâmetro, para assim já definir em sua declaração seu comportamento.*

*Adaptando a função repassada como base, replicada aqui nas duas primeiras linhas de nosso código apenas como referência, vamos buscar a reescrever de forma totalmente discriminada.*

*Inicialmente usamos o marcador def para que o interpretador imediatamente entenda que a linha / bloco de código subsequente se refere a uma função. Seguindo o exemplo, nomeamos tal função como inventario( ).*

*Na sequência, dentro do campo destinado aos parâmetros da função, inserimos o primeiro parâmetro item, nesse contexto um parâmetro comum, seguido de localização, que aqui de acordo com sua sintaxe é um parâmetro nomeado e justaposto de valor padrão definido. Em seguida é definido o parâmetro quantidade, aqui possuindo um valor pré-definido, seguido de preco, que aqui graças a notação ": float", possui tipo de dado e valor definidos manualmente. Também é criado um parâmetro chamado categoria, aqui baseado em um contêiner de dados.*

*Finalizando, temos a inserção de um campo **kwargs que abre instância para que mais parâmetros personalizados sejam inseridos caso necessário.*

*Por fim, a notação "-> np.ndarray" configura que os parâmetros da função / método de classe inventario( ) serão especificamente do tipo array Numpy.*

*Dessa forma, cumprimos todos os requisitos da questão. Não havendo nenhum erro de sintaxe, ao executar este bloco de código nenhum retorno direto será gerado, pois no corpo de função nenhum retorno específico foi definido.*

---
---

**13 – Crie uma classe composta por atributos “somente leitura”, da forma mais reduzida possível:**

```
from dataclasses import dataclass

@dataclass(frozen=True)
class Cachorro:
    cor: str
    idade: int

pet1 = Cachorro('Preto e Branco', 6)

print(pet1.cor)
```

*Para criação de uma classe que, após utilizada uma vez, reserva seus objetos e atributos de classe de modo a ficarem instanciáveis porém imutáveis, existe uma vasta gama de possibilidades. Para realizarmos tal feito, da forma mais reduzida possível conforme especificado pela questão, podemos fazer uso de dataclasses.*

*Como a estrutura de código de uma dataclass não vem por padrão importada pelo núcleo Python, é necessário do pacote dataclasses importar o marcador dataclass.*

*Na sequência, diretamente no escopo global do código, inserimos o marcador @dataclass( ) aqui tendo seu parâmetro frozen configurado para True, para que de fato os objetos sobre o escopo desse marcador sejam imutáveis após sua criação.*

*Sendo assim, criamos uma classe de nome Cachorro, dentro de seu corpo criamos apenas duas instâncias, sendo a primeira de nome cor, que receberá algum dado / valor obrigatoriamente em formato de string.*

*Da mesma forma criamos uma instância chamada idade, que receberá como dado / valor um número do tipo int.*

*Uma vez terminada a escrita da classe, podemos testar a funcionalidade da mesma. Para isso, de volta ao escopo geral do código criamos uma variável de nome pet1, que cria uma instância da classe Cachorro( ), alimentando a mesma com 'Preto e Branco' para cor e 6 para idade.*

Apenas para fins de exemplo, repassando como parâmetro para função print( ) os últimos dados atribuídos para pet1.cor, nos é retornado:

**Preto e Branco**

```
pet1.cor = 'Preto com pintas brancas'
```

*Instanciando o objeto pet1.cor, ao tentar realizar qualquer tipo de alteração sobre seu conteúdo será gerado um erro do tipo FrozenInstanceError, confirmando assim que nesse contexto os dados iniciais são imutáveis.*

**---------------------------------------------------------**
**FrozenInstanceError  Traceback (most recent call last)**
**<ipython-input-3-1009e962a79b> in <module>()**
**12 print(pet1.cor)**
**13**
**---> 14 pet1.cor = 'Preto com pintas brancas'**
**15**
**<string> in __setattr__(self, name, value)**

**FrozenInstanceError: cannot assign to field 'cor'**

**----------**

**14 – Crie manualmente um sistema de contagem regressiva, a contar de 10, atualizando seus dados a cada 2 segundos.**

```
import time

final = 0

total = 10

print('Contagem iniciada...')

while final <= 10:
    print(total)
    time.sleep(2)
    total -= 1
    final = total
    if total == -1:
        print('Contagem encerrada!!!')
        break
```

*Como mais uma vez estamos a manipular dados temporais, vamos novamente recorrer a algumas funcionalidades da biblioteca time.*

*Sendo assim, logo na primeira linha importamos todas as ferramentas da biblioteca time através de um simples import.*

*Em seguida criamos duas variáveis de controle, nomeadas como final e total, respectivamente que guardam valores numéricos os quais faremos uso logo a frente.*

*Como uma contagem regressiva é, internamente, a execução de uma determinada tarefa repetidas vezes, podemos fazer esse ciclo de processamento através de uma estrutura de repetição while.*

*Dessa forma, criamos uma simples estrutura condicional a ser repetida quantas vezes forem necessário até validar algum critério*

*definido. Nesse caso, enquanto o valor de final for igual ou menor que 10, o ciclo de repete.*

*Indentado a esta estrutura condicional / de repetição inicial, exibimos em tela o último valor atribuído para variável total.*

*Em seguida, para criar o mecanismo que atualizará a contagem regressiva de 2 em 2 segundos, instanciamos e inicializamos o método time.sleep( ), por sua vez parametrizado com 2.*

*Logo após, de acordo com a ordem de leitura léxica do interpretador, a variável total é instanciada novamente, tendo seu valor agora atualizado, decrementado em uma unidade. Nos mesmos moldes, a variável final é instanciada, tendo seu último valor atribuído atualizado com o último valor atribuído para a variável total.*

*Dessa forma, estamos criando uma regra que as variáveis terão seus devidos valores atualizados a cada 2 segundos por um novo ciclo de repetição.*

*Por fim, é criada uma simples validação onde, se o valor da variável total foi igual a -1, a contagem regressiva é encerrada, pois de acordo com o enunciado da questão, a mesma deve ser feita iniciando em 10, terminando em 0.*

Não havendo nenhum erro de sintaxe, o retorno será:

**Contagem iniciada...**
**10**
**9**
**8**
**7**
**6**
**5**
**4**
**3**
**2**
**1**

**0**
**Contagem encerrada!!!**

**----------**

**15 – Crie uma estrutura de código que simula o sistema de dependência de uma blockchain em um cluster de processamento. Nesse processo é permitido usar de alguma estrutura de repetição, e o cluster inicial deve conter ao menos 4 blocos de origem.**

```
def executa(base_dir, no_dir, chain):
  caminhos = []
  base = base_dir
  no = no_dir
  chain = chain
  caminhos.append[0](base)
  caminhos.append[1](no)
  caminhos.append[2](chain)
  return caminhos

class Cluster:
  def __init__(self, bloco, params, configs):
    self.bloco = bloco
    self.params = params
    self.configs = configs
    self.nodes = ['bloco_01', 'bloco_02', 'bloco_03', 'bloco_04']
    self.cria_bloco = {node: executa(caminho = caminhos,
                              bloco = self.bloco,
                              params = self.params,
                              configs = self.configs) for node in self.nodes}
```

*Este exercício pode ser uma dor de cabeça para quem não está familiarizado a arquitetura de uma blockchain. Por hora, raciocine que uma blockchain é um sistema de rede distribuída onde cada nó da rede é um bloco de processamento, bloco este interligado de modo que uma determinada tarefa é distribuída por todos os nós da rede, de forma que através de um contrato*

*inteligente, cada nó colabora de sua forma com o processamento, sendo validado e recompensado por sua participação sempre que a tarefa termina.*

*Sendo assim, o convencional é que cada um desses nós da rede seja uma classe identificada pela rede com suas respectivas estruturas internas (privadas), tendo a usabilidade de suas funções / métodos de classe compartilhadas para um cluster de processamento.*

*Para que haja essa comunicação entre os nós, de modo a organizar a distribuição das tarefas e a validação da parcela de contribuição de cada parte, é criado o chamado sistema de dependência, que "força" a atualização de todos os nós da rede constantemente, para que não haja discrepância dos dados compartilhados.*

*Dito isto, vamos buscar resolver o problema proposto por essa questão.*

*Logo de início, é criada uma função de nome executa( ), que recebe obrigatoriamente dados para os parâmetros base_dir, no_dir e chain.*

*Dentro do corpo desta função são declaradas algumas variáveis, sendo a primeira delas nomeada como caminhos, que recebe como atributo inicialmente uma lista vazia; base que receberá o dado repassado pelo parâmetro base_dir; no que receberá o dado repassado pelo parâmetro no_dir e chain, que da mesma forma como feito anteriormente, recebe como atributo o dado repassado pelo parâmetro chain.*

*Em seguida é instanciada a variável caminhos, agora inserindo dentro da lista associada a mesma via método append( ), em sua posição 0 de índice, o último dado atribuído para base.*

*Da mesma forma na posição de índice 1 de caminhos é inserido o último dado lido na variável no, por fim para posição de índice 2 da variável caminhos é inserido o último dado extraído da variável chain.*

*Por fim, é gerado como retorno o último estado da variável caminhos.*

*Na sequência, vamos criar a classe responsável pelo cluster inicial da rede, nesse caso, criando uma classe de nome Cluster, que possui indentado para si um método construtor __init__( ) que além de fazer instância a própria classe, recebe como objetos a serem repassados como atributos de classe um bloco, params e configs.*

*Criando as devidas instâncias, inicialmente atribuímos para self.bloco os dados repassados por bloco, o mesmo processo é feito para self.params e self.configs.*

*Logo após é criado um novo objeto atributo de classe chamado self.nodes, que recebe em formato de lista os nomes a serem utilizados para os 4 primeiros blocos.*

*Por fim, é criado um último objeto / atributo de classe chamado self.cria_bloco que por sua vez, de fato cria o mecanismo para o sistema de dependência, usando de dictionary comprehension para percorrer cada um dos elementos de nodes, aplicando sobre os mesmos o método de classe executa( ), gerando assim uma instância para cada.*

*Desse modo, assim que executado este código as instâncias iniciais serão geradas e a cada iteração realizada com Cluster todas as instâncias são devidamente atualizadas.*

*Executar este código neste momento não gera nenhuma saída explícita, a menos que no escopo geral do código alguma variável instancie a classe Cluster, utilizando da estrutura da mesma.*

---

**16 – Escreva um programa que, a partir de uma lista de elementos, retorne o tipo de dado de cada elemento. A lista em questão é: lista=[(1,),[1,2,3],(1),{4,5},{'nome':'A','value':2},10, [],1+3j,1.2,True,False,None,'Hello World!'].**

```
lista=[
      (1,),
      [1,2,3],
      (1),
      {4,5},
      {'nome':'A','value':2},
      10,
      [],
      1+3j,
      1.2,
      True,
      False,
      None,
      'Hello World!',
   ]

for elemento in lista:
  print(elemento,type(elemento))
```

*Para inspecionar o tipo de dado específico de um objeto em Python, podemos fazer o uso do método type( ), por sua vez parametrizado com o objeto / variável o qual devemos retornar o tipo de dado.*

*Sendo assim, inicialmente replicamos a lista proposta pela questão, inclusive respeitando a sintaxe de cada elemento da lista.*

*Na sequência, através de um laço de repetição for percorremos cada um dos elementos da lista, e a cada ciclo de repetição repassamos como parâmetro para nossa função print( ) o*

*elemento em si através da instância de sua variável, seguido do método type( ), aqui parametrizado com as mesma variável.*

*Dessa forma, iremos gerar como saída / retorno a lista de cada um dos elementos mapeados assim como seu tipo.*

Nesse caso, o retorno será:

**(1,) <class 'tuple'>**
**[1, 2, 3] <class 'list'>**
**1 <class 'int'>**
**{4, 5} <class 'set'>**
**{'nome': 'A', 'value': 2} <class 'dict'>**
**10 <class 'int'>**
**[] <class 'list'>**
**(1+3j) <class 'complex'>**
**1.2 <class 'float'>**
**True <class 'bool'>**
**False <class 'bool'>**
**None <class 'NoneType'>**
**Hello World! <class 'str'>**

**----------**

**17 – Crie uma simples ferramenta de análise exploratória de dados, que recebe uma lista / array, retornando a partir dos dados da mesma os valores referentes ao maior número, menor número, número de elementos totais, soma de todos elementos, média aritmética, mediana, variança e desvio padrão. A lista em questão é: lista = [-1505, 2518, 2333, 4682, -1740, 1067, 995, 22, 2, -1153, -4098, 4560, 2958, 3640, -4154, 2345,4, -1601, 158, -4044, -98, 707, 359, -3088, 527, -3004, -4045, 563, -4137, 4598, -3862, 488, -1, 3445, 3820, 504, -1475, 1626, -1252, 2059, 16, -1472, 2610, -4643, 2912, -2517, -4604, -1476, 3950, -4640, -229, 229, -3452, 4309, 2356, 66, 3728, -1846, -635, -3581, 4457, -2592, -1066,4, -1006, -164, 805, -4500, -455, 2245, -4959,2, -2415, 2038, 4512, 1243, 349, -1153, 3623, 631, 2209, -3349, -2417, 4429, -1324, -4101, -1354, 4400, -4968, -4433, 2042, 904, 932, 1331, 4955, -3775, -333, 1012, 2192, -161, -224, 1505, -1615, -2165, 3666, -4253, -358, -3939, -2641, 1228, -608, -2280, 4939, -3355, 1340, -57, 3346, 2686, -1572, 1991, -2351, -3972, -1683, -1509, -2488, 266, -2991, -795, -4032, 2397, 2391, 3654, -1044, -2204, -2440, -1280, 2714, -4151, -1951, 3684, -4251, 3748, -4359, -1436, -2190, 4538, -3563, 1542, 1926, -3940, -2274, -4223, 2504, -4112, 2388]**

```
from statistics import median, variance, stdev

lista=[-1505, 2518, 2333, 4682, -1740, 1067, 995,22,2,
       -1153, -4098, 4560, 2958, 3640, -4154, 2345,4,
       -1601, 158, -4044, -98, 707, 359, -3088, 527,
       -3004, -4045, 563, -4137, 4598, -3862, 488, -1,
```

```
        3445, 3820, 504, -1475, 1626, -1252, 2059, 16,
        -1472, 2610, -4643, 2912, -2517, -4604, -1476,
        3950, -4640, -229, 229, -3452, 4309, 2356, 66,
        3728, -1846, -635, -3581, 4457, -2592, -1066,4,
        -1006, -164, 805, -4500, -455, 2245, -4959,2,
        -2415, 2038, 4512, 1243, 349, -1153, 3623, 631,
        2209, -3349, -2417, 4429, -1324, -4101, -1354,
        4400, -4968, -4433, 2042, 904, 932, 1331, 4955,
        -3775, -333, 1012, 2192, -161, -224, 1505, -1615,
        -2165, 3666, -4253, -358, -3939, -2641, 1228,
        -608, -2280, 4939, -3355, 1340, -57, 3346, 2686,
        -1572, 1991, -2351, -3972, -1683, -1509, -2488,
        266, -2991, -795, -4032, 2397, 2391, 3654, -1044,
        -2204, -2440, -1280, 2714, -4151, -1951, 3684,
        -4251, 3748, -4359, -1436, -2190, 4538, -3563,
        1542, 1926, -3940, -2274, -4223, 2504, -4112, 2388]

print(f'Maior número: {max(lista)}')
print(f'Menor número: {min(lista)}')
print(f'Soma de todos os elementos da lista: {sum(lista)}')
print(f'Quantidade de elementos da lista: {len(lista)}')
print(f'Média aritmética: {sum(lista)/len(lista):.2f}')
print(f'Mediana: {median(lista)}')
print(f'Variância: {variance(lista):.2f}')
print(f'Desvio padrão: {stdev(lista):.2f}')
```

*Para darmos início a resolução deste problema, inicialmente importamos da biblioteca nativa statistics as ferramentas median, variance e stdev, que automatizarão o processe de extração de dados referentes a mediana, variança e desvio padrão, respectivamente. Isto deve ser suficiente pois para retornarmos as demais informações podemos fazer uso de métodos inclusos nos built-ins do Python.*

*Logo após replicamos a lista descrita no enunciado da questão e, feito isto, estamos prontos para gerar / produzir os dados solicitados pela questão.*

*Logicamente existem muitas formas de se gerar tal tipo de retorno, pensando na questão de praticidade, podemos retornar os devidos dados fazendo uso de métodos inseridos dentro das máscaras de substituição de uma f'string, diretamente como parâmetro para nossa função print( ).*

*Sendo assim, criamos uma primeira função print( ) que exibe em tela o elemento de maior valor numérico mapeado na lista através do método max( ). Repetimos exatamente o mesmo processo para o menor número encontrado na lista por meio do método min( ); Usando do método sum( ) realizamos a soma de todos os elementos da lista; Através do método len( ) retornamos o tamanho / número de elementos da lista; Dividindo a soma de todos os elementos da lista pelo seu tamanho via sum( ) / len( ) temos como retorno a média aritmética; Utilizando do método median( ), importado anteriormente, realizamos o cálculo da mediana; Nos mesmos moldes utilizando de variance( ) e stdev( ) obtemos dados referentes a variança e ao desvio padrão, respectivamente.*

Nesse caso, o retorno será:

**Maior número: 4955**
**Menor número: -4968**
**Soma de todos os elementos da lista: -38259**
**Quantidade de elementos da lista: 155**
**Média aritmética: -246.83**
**Mediana: -161**
**Variância: 7629024.95**
**Desvio padrão: 2762.07**

**----------**

**18 – A partir da lista lista=['a', 'a', 'a', 'b', 'c', 'k', 'a', 1, 2, 3, 4, 'j', 'l', 'm'] gere uma nova lista porém sem elementos repetidos.**

```
lista=[
      'a','a','a',
      'b','c','k',
      'a',1,2,3,4,
      'j','l','m',
   ]

lista2=list(set(lista))

print(lista)
```

```
print(lista2)
```

*Usando de simples lógica, de acordo com os tipos de dados básicos em Python, ao mesmo tempo que temos tipos de dados como listas e dicionários que permitem a duplicidade de dados, temos tipos de dados específicos que não permitem tal duplicidade.*

*Sendo assim, da forma mais reduzida possível, se simplesmente convertermos uma lista para um set, no processo de conversão automaticamente os dados / valores / elementos repetidos serão eliminados.*

*Direto ao código, reproduzida a lista proposta pela questão, podemos partir para a geração da nova lista.*

*Para isso, declaramos uma nova variável de nome lista1, que recebe como seu atributo em forma de lista via list( ) o retorno do método set( ), por sua vez parametrizado com a variável lista.*

Feita a conversão, ao exibir em tela via print( ) o conteúdo da variável lista2, o retorno será:

**['a', 'a', 'a', 'b', 'c', 'k', 'a', 1, 2, 3, 4, 'j', 'l', 'm']**
**[1, 2, 3, 4, 'l', 'c', 'm', 'j', 'a', 'k', 'b']**

---

**19 – Supondo que temos um simples carrinho de compras, em formato de lista composta por alguns elementos onde cada elemento da mesma é um dicionário descrevendo um item, crie um mecanismo que retorna ao usuário o valor total do carrinho, seu item mais caro e também seu item mais barato.**

```
lista_compras = [{'nome':'feijão', 'preco':9.79},
                 {'nome':'arroz', 'preco':3.45},
                 {'nome':'carne', 'preco':20.93},
                 {'nome':'macarrão', 'preco':2.99}]

soma = sum([produto['preco'] for produto in lista_compras])

print(f'A cesta de produtos custa: R$ {soma:.2f}')

produto_max = max(lista_compras,key = lambda produto:produto['preco'])
produto_min = min(lista_compras,key = lambda produto:produto['preco'])

print(f"Produto mais caro: {produto_max['nome'].title()} - R$ {produto_max['preco']:.2f}" )
print(f"Produto mais barato: {produto_min['nome'].title()} - R$ {produto_min['preco']:.2f}")
```

*Embora o enunciado da questão pareça estar pedindo algo relativamente complexo, na verdade estaremos utilizando apenas de 3 métodos para conseguir extrair as informações necessárias solicitadas pela questão.*

*Inicialmente é declarada uma variável de nome lista_compras, que como atributo recebe uma lista composta de alguns elementos em forma de dicionário, conforme descrito no enunciado da questão. Para cada elemento da lista são criadas duas chaves com seus respectivos valores, simulando itens de mercado com seus respectivos preços.*

*Logo após criamos uma variável de nome soma, que como atributo recebe o retorno da função sum( ), aqui parametrizado com um dictionary comprehension que usando o laço de repetição for percorre de cada elemento de lista_compras, mais especificamente onde constar algum dado/valor para a chave 'preco', retornando esse valor para a variável temporária produto. Desse modo, a cada ciclo de repetição um valor é mapeado e somado ao próximo valor do próximo ciclo, até que todos os valores tenham sido iterados.*

*De modo muito parecido, criamos duas novas variáveis de nomes produto_max e produto_min, que usando dos métodos max( ) e min( ), parametrizados com um dictionary comprehension associado a uma função anônima, extrai de produto o item de maior e de menor valor atrelado ao campo 'preco', respectivamente.*

*Feito isso, podemos simplesmente exibir em tela os resultados obtidos através da função print( ), nesse caso, fazendo uso de f'strings onde dentro das máscaras de substituição são instanciados o nome do item (convertendo a primeira letra de seu nome em maiúsculo via title( )), seguido de seu preço aqui formatado com duas casas decimais após a vírgula apenas por convenção.*

O retorno será:

**A cesta de produtos custa: R$ 37.16**
**Produto mais caro: Carne - R$ 20.93**
**Produto mais barato: Macarrão - R$ 2.99**

---

**20 – Uma vez apresentada a lista x = [10, 12, 35], apresente ao menos 5 formas de retornar ao usuário a soma dos elementos dessa lista.**

```
x = [10, 12, 35]

# Método 1
num = x[0] + x[1] + x[2]
print(num)

# Método 2
num = sum([i for i in x])
print(num)

# Método 3
num = sum(x)
print(num)

# Método 4
def soma(x):
    total = 0
    for i in x:
        total +=i
    print(total)
    return total
num = soma(x)

# Método 5
n1, n2, n3 = x
num = n1 + n2 + n3
print(num)
```

*Aqui temos um exercício bem característico da linguagem Python, pois como bem sabemos, a mesma é muito dinâmica,*

*possibilitando uma vasta gama de possibilidade de formas de se realizar a mesma tarefa. No exercício proposto é pedido 5 formas de retornar a soma dos elementos de uma lista, certamente devem existir ao menos 10 outras formas de solucionar o mesmo problema, vamos nos ater as formas mais comumente utilizadas, sem ordem específica, apenas explorando suas características.*

*Inicialmente replicamos a lista proposta pelo exercício, atribuindo seu conteúdo para uma variável de nome x.*

*Como primeiro método, podemos extrair elementos específicos da lista de acordo com seu número de índice e, em função de seu tipo de dado ser numérico, realizar diretamente a soma destes elementos na atribuição.*

*Para isso, declaramos uma variável de nome num que recebe como atributo a soma dos elementos de índice 0, 1 e 2, respectivamente, da lista atribuída a x.*

*Em seguida exibimos em tela o conteúdo de num através da função print( ).*

*Como segundo método vamos buscar fazer uso do método embutido sum( ), atribuído a variável num, repassando como parâmetro para o mesmo um laço de repetição que a cada ciclo extrai um dos elementos da lista. Dessa forma, a cada ciclo de repetição um dos elementos é mapeado e extraído, sendo somado com o elemento mapeado e extraído pelo próximo ciclo de repetição.*

*Por fim, exibimos novamente o conteúdo de num via print( ).*

*Como terceiro método podemos simplificar o método anterior, criando uma variável de nome num, que instancia e inicializa o método sum( ), repassando como parâmetro para o mesmo o conteúdo da variável x, que por ser composta apenas de elementos numéricos, terá cada um de seus elementos somados diretamente.*

*Mais uma vez, exibimos em tela o conteúdo de num via função print( ).*

*Como quarto método podemos transliterar o processo de soma em uma função personalizada.*

*Para isso inicialmente definimos uma função de nome soma( ) que obrigatoriamente recebe um elemento x. Dentro do corpo dessa função criamos uma variável de controle chamada total, inicialmente de valor 0.*

*Na sequência é criado um laço de repetição que percorre o conteúdo da variável x, retornando a cada ciclo de repetição um dado / valor para a variável temporária i. Indentado ao laço de repetição, instanciamos a variável total, atualizando seu conteúdo com a soma de seu valor atual com o último valor lido na variável temporária i.*

*Ainda dentro do laço de repetição inserimos uma função print( ) para exibir em tela o último valor atribuído para a variável total, finalizando com o próprio retorno da função que nesse caso retorna o último dado / valor atribuído a variável total.*

*De volta ao corpo geral do código é criada a variável num, que instancia e inicializa nossa função soma( ), repassando como parâmetro para a mesma a variável x. Assim, como atributo para num teremos o resultado da soma dos elementos de x.*

*Por fim, como quinto método, podemos usar da soma simples de valores de variáveis.*

*Para isso, logo de início declaramos três variáveis de nomes n1, n2 e n3, respectivamente, que recebem como atributo o conteúdo de x. Nesse caso, de acordo com a forma da declaração e atribuição dessas variáveis, o que vai ocorrer é o desempacotamento dos três elementos da variável x para as 3 variáveis em questão.*

*Na sequência criamos uma variável de nome num que recebe como atributo o resultado da soma das variáveis n1, n2 e n3.*

*Por fim, mais uma vez exibimos em tela o conteúdo de num via print( ).*

Não havendo nenhum erro de sintaxe o retorno será:

**57**
**57**
**57**
**57**
**57**

**----------**

**21 – Uma prática comum é para certos contextos gerar listas, dicionários, ou qualquer outro tipo de contêiner de dados com alguns valores padrão ou gerados a partir de algum critério. Nessa linha de raciocínio, crie um gerador de dicionário que, com um dado inteiro n fornecido pelo usuário, retorna um dicionário composto por todos números antecessores ao número n e seus valores quando elevados ao quadrado.**

```
n = int(input('Digite um número: '))
```

```
d = dict()
for i in range(1, n + 1):
    d[i] = i * i

print(d)
```

*Direto ao ponto, na primeira linha criamos uma variável n que através do método input( ) recebe do usuário um número, validando o mesmo como inteiro caso inicialmente não seja.*

*Em seguida declaramos uma nova variável, dessa vez de nome d, que instancia o método dict( ), sem parâmetros mesmo, para que possamos usado do mesmo como um criador / gerador de dicionários.*

*Na sequência criamos um laço de repetição for que percorre toda a extensão dos números até chegar em n, de uma em uma unidade.*

*Indentado dentro desse laço instanciamos a variável d, na posição i para que sejam criadas as chaves referentes a cada unidade de ciclo de repetição, atribuindo como valores para tais chaves o valor de i multiplicado por si próprio para que assim o número seja elevado ao quadrado.*

*Por fim simplesmente exibimos em tela via print( ) o conteúdo do dicionário gerado para d.*

Nesse caso, supondo que o usuário tenha digitado 12, o retorno será:

**Digite um número: 12**
**{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11: 121, 12: 144}**

**----------**

**22 – Crie um diretório chamado backups, dentro desse diretório, crie uma pasta numerada para cada mês do ano, por exemplo mês_03. Antes de criar a pasta backup, verifique se a mesma já existe no diretório raiz do projeto.**

```
import os

if not os.path.exists('backups'):
  os.mkdir('backups')

  dir = [f'mês_{str(i).zfill(2)}' for i in range(1, 13)]
  for i in dir:
      path = os.path.join('backups', i)
      os.mkdir(path)
  print(sorted(os.listdir('backups')))

else:
  print('Pasta já existente')
```

*Sempre que estamos trabalhando de modo a interagir com estruturas internas do sistema operacional, é comum fazer uso das ferramentas disponíveis na biblioteca os do núcleo Python para automatizar certos procedimentos.*

*Sendo assim, inicialmente importamos a biblioteca os através do comando import os.*

*Em seguida já podemos partir para a validação que verificará se a pasta a qual queremos criar já existe. Nesse processo, podemos combinar uma negação lógica com o método os.path.exists( ) para assim realizar a validação.*

*Em nosso código, criamos uma estrutura condicional que caso não exista uma pasta de nome 'backups', a mesma é criada através do método os.mkdir( ).*

*Uma vez criada a pasta base, podemos partir para criação das demais pastas, referentes aos meses conforme solicitado na questão.*

*Para isso, ainda indentado a estrutura condicional criada anteriormente, declaramos uma nova variável de nome dir, que recebe em forma de lista o retorno de um laço de repetição que percorre um intervalo de 1 a 13 (lembrando que em Python, se tratando de extensão de uma lista, o valor do último elemento é o gatilho final para o tamanho, nesse caso, 13 para que 12 seja o último elemento mapeado), acrescentando como prefixo o texto 'mês_' seguido do número do ciclo de repetição em questão convertido para string. Dessa forma, atribuído para a variável dir teremos uma lista composta dos nomes das pastas a serem criadas (mês_01, mês_02, e assim por diante).*

*Próximo passo, vamos para a criação de fato das pastas. O processo pode ser feito de diversas formas, aqui, mantendo o uso de funções pré-moldadas da biblioteca os, vamos buscar gerar as devidas pastas a partir da combinação de alguns métodos.*

*Nesse caso, criamos um laço de repetição que percorre cada um dos elementos de dir, sendo que a cada ciclo de repetição, com base no dado / valor atribuído para a variável i, estaremos criando sua respectiva pasta.*

*Para isso, indentado ao laço de repetição for, criamos uma variável de nome path que instancia e inicializa o método os.path.join( ), aqui parametrizado com o nome da pasta raiz e com o nome da pasta instanciada em i.*

*Diretamente no mesmo escopo inicializamos o método os.mkdir( ) por sua vez parametrizado com path para que assim de fato seja criada a pasta de acordo com os dados lidos dentro deste ciclo de repetição.*

*Por fim exibimos em tela o conteúdo final, de todas as pastas criadas, repassando como parâmetro para função print( ) os métodos sorted( ) e os.listdir( ).*

*Finalizando, de volta ao escopo global do código podemos dar um destino final para a estrutura condicional criada anteriormente, aqui para fins de encerrar a validação executada em busca de, se a pasta backups já existir, nada precisará ser feito. Eis que assim resolvemos a questão de uma forma razoavelmente simples e reduzida.*

Executando esse código o retorno será:

**['mês_01', 'mês_02', 'mês_03', 'mês_04', 'mês_05', 'mês_06', 'mês_07', 'mês_08', 'mês_09', 'mês_10', 'mês_11', 'mês_12']**

Executando mais uma vez, o retorno será:

**Pasta já existente**

---

**23 – Crie um simples jogo de adivinhação que gera um número aleatório de 0 a 10, em seguida recebe um número do usuário e, caso o mesmo tenha acertado o número, recebe uma mensagem de sucesso, caso contrário, pede que o usuário insira um novo número até acertar a adivinhação.**

```
import random

num_gerado, num_adivinhado = random.randint(0, 11), 0

while num_adivinhado != num_gerado:
    num_adivinhado = int(input('Digite um número entre 0 e 10: '))

print('Parabéns, você adivinhou o número!!!')

print(f'Número gerado: {num_gerado}')
```

*Se tratando da geração de números aleatórios, o uso da biblioteca random é sempre bem vindo pois a mesma possui uma série de métodos os quais podemos facilmente implementar em nossos códigos para geração.*

*Para o exercício proposto, precisamos gerar um número aleatório dentro de um intervalo definido, em seguida precisamos validar um número digitado pelo usuário, verificando se possui mesmo valor.*

*A forma mais reduzida de se contornar esse problema é usando de um gerador de número aleatório associado a uma estrutura de repetição.*

*Sendo assim, na primeira linha de nosso código importamos a biblioteca random, em seguida declaramos duas variáveis de nomes num_gerado e num_adivinhado, que de acordo com a forma de declaração do código, a primeira variável instancia e inicializa o*

*método random.randint( ), por sua vez parametrizado com 0 e 11, seguido do atributo da segunda variável, que inicialmente é simplesmente o valor 0, a ser atualizado.*

*Logo após criamos uma estrutura condicional while, que realiza uma validação onde enquanto o valor de num_adivinhado for diferente do valor num_gerado, solicita ao usuário que digite um novo número instanciando a variável num_adivinhado e inicializando a partir da mesma o método input( ).*

*Por fim, simplesmente criamos os retornos via função print( ) para assim interagir com o usuário.*

Executando o código é iniciada a interação com o usuário:

**Digite um número entre 0 e 10: 5**
**Digite um número entre 0 e 10: 4**
**Digite um número entre 0 e 10: 8**
**Digite um número entre 0 e 10: 7**
**Digite um número entre 0 e 10: 2**
**Digite um número entre 0 e 10: 1**
**Digite um número entre 0 e 10: 10**
**Digite um número entre 0 e 10: 3**
**Digite um número entre 0 e 10: 6**
**Digite um número entre 0 e 10: 9**
**Parabéns, você adivinhou o número!!!**
**Número gerado: 9**

---

**24 – Crie um template de mala direta contendo uma mensagem, envie a mensagem para 5 pessoas listadas em uma simples base de dados aqui representada pela lista nomes = ['Anna', 'Paulo', 'Maria', 'Rafael', 'Patricia'] de modo que a mensagem seja personalizada a cada um dos nomes da lista.**

```
from string import Template

nomes = ['Anna', 'Paulo', 'Maria', 'Rafael', 'Patricia']

email = """Olá $name,

Seja muito bem vindo(a) ao curso Python!!!

Abraço, Fernando Feltrin"""

template = Template(email)

for i in nomes:
    print(template.substitute(name = i))
    print('-' * 42)
```

*Em Python existe uma grande variedade de formas como podemos lidar com templates / blueprints ou qualquer outro tipo de "estrutura molde".*

*Para nosso problema atual, onde teremos que criar uma mensagem a ser encaminhada para 5 pessoas, sendo exatamente a mesma mensagem para todas, com o diferencial da personalização, vamos buscar a implementação desta estrutura de código da forma mais direta o possível. Para isso, usaremos da ferramenta Template da biblioteca string.*

*Sendo assim, logo de início realizamos a importação de Template, seguido da declaração da variável nomes que aqui recebe uma lista de 5 nomes conforme o enunciado da questão.*

*Na sequência partimos diretamente para elaboração da mensagem, declarando uma nova variável de nome email, que recebe em forma de comentário um texto.*

*Repare que logo na primeira linha do texto é inserido um marcador "$" seguido de um nome, nesse caso name. Pois bem, é exatamente iterando sobre este marcador que estaremos inserindo os devidos nomes personalizados para as mensagens.*

*Em seguida criamos uma nova variável de nome template, que instancia e inicializa o método Template( ) parametrizando o mesmo com o conteúdo de email. Até aqui nada de novo, então podemos partir para a personalização do template.*

*É então criado um laço de repetição for, que percorrerá todos os elementos de nomes, retornando a cada ciclo de repetição um dado / valor para variável temporária i.*

*Indentado ao corpo deste laço de repetição, inserimos diretamente duas funções print( ), sendo que como parâmetro para a primeira repassamos o método substitute( ), aplicado sobre template, buscando o marcador definido anteriormente name e substituindo seu dado / valor pelo último dado atribuído a i. Por fim, o segundo print( ) apenas cria uma linha divisória para que possamos ver de forma mais legível o retorno deste código.*

Sendo assim, não havendo nenhum erro de sintaxe o retorno será:

**Olá Anna,**

**Seja muito bem vindo(a) ao curso Python!!!**

**Abraço, Fernando Feltrin**

**----------**

**Olá Paulo,**

**Seja muito bem vindo(a) ao curso Python!!!**

**Abraço, Fernando Feltrin**

**----------**

**Olá Maria,**

**Seja muito bem vindo(a) ao curso Python!!!**

**Abraço, Fernando Feltrin**

**----------**

**Olá Rafael,**

**Seja muito bem vindo(a) ao curso Python!!!**

**Abraço, Fernando Feltrin**

**----------**

**Olá Patricia,**

**Seja muito bem vindo(a) ao curso Python!!!**

**Abraço, Fernando Feltrin**

**----------**

**----------**

**25 – Crie um simples mecanismo que lê um determinado texto, nesse caso “Python 3.9”, retornando ao usuário a quantidade de números encontrados no texto e que números são estes, ordenados em ordem posicional (na exata ordem em que foram identificados no texto).**

```
import re

texto = 'Python 3.9'

numeros = re.findall(pattern=r'\d', string=texto)

print(f'Foram encontrados {len(numeros)} números no texto "{texto}".')
print(f'Os números são {numeros[0]} e {numeros[1]}, respectivamente.')
```

*Por mais que o enunciado tenha nos “vendido dificuldade”, o problema o qual estamos a tratar pode ser facilmente resolvido usando de expressões regulares, já que se trata da identificação de caracteres numéricos em meio a um texto.*

*Partindo para prática, importamos a biblioteca re para usar de suas ferramentas, em seguida criamos uma variável de nome texto que recebe como atributo a string ‘Python 3.9’.*

*Na sequência declaramos uma nova variável de nome numeros, que por sua vez instancia e inicializa o método re.findall( ), por sua vez parametrizado com o padrão de caractere a ser mapeado, nesse caso numérico, seguido da variável que possui o texto de origem.*

*Uma vez concluída a escrita das estruturas fundamentais, podemos exibir em tela os devidos resultados.*

*É criada uma função print( ), que fazendo uso de f’strings e suas respectivas máscaras de substituição, instancia no texto o*

*“tamanho” da variável numeros e o texto de origem.*

*Por fim, dentro de uma nova função print( ), em sua f’string, instanciamos os elementos de posição de índice 0 e 1, respectivamente, para assim exibir em tela ao usuário tais elementos na ordem solicitada.*

Neste caso, o retorno será:

**Foram encontrados 2 números no texto "Python 3.9".**
**Os números são 3 e 9, respectivamente.**

---

**26 – Crie um simples mecanismo validador que recebe do usuário um texto qualquer e retorna ao mesmo se tal texto pertence ou não a uma frase pré-definida, nesse caso: 'Python é uma excelente linguagem de programação!!!'**

```
texto = 'Python é uma excelente linguagem de programação!!!'

pesquisa = input('Digite uma palavra a ser pesquisada: ')

if (texto.find(pesquisa) == -1):
    print(f'Palavra "{pesquisa}" não encontrada no texto de origem.')
else:
    print(f'Palavra "{pesquisa}" é uma das palavras do texto de origem')
```

*Para realizarmos uma validação simples, verificando se uma determinada palavra pertence a um texto, podemos simplesmente realizar tal verificação combinando uma estrutura condicional com o método find( ).*

*A nível de código, inicialmente declaramos uma variável de nome texto, que recebe como atributo em forma de string o texto proposto pela questão.*

*Logo após declaramos uma nova variável, dessa vez de nome pesquisa, que através do método input( ) recebe do usuário algum texto digitado pelo mesmo.*

*Criadas as variáveis as quais estaremos fazendo uso, podemos partir para elaboração da estrutura condicional.*

*Aqui, como o problema proposto baseia-se em encontrar um certo conjunto de caracteres em meio a outro conjunto, criamos uma estrutura condicional que verifica, instanciamos o conteúdo de texto aplicando sobre o mesmo o método find( ), aqui parametrizado com o conteúdo de pesquisa, usando da notação == -1 para que tal verificação seja feita em todos os caracteres do texto de origem.*

*Por fim, caso esta proposição seja verdadeira é exibido ao usuário uma mensagem de sucesso, caso contrário, uma mensagem explicando que a palavra inserida não compõe a estrutura do texto base.*

Simulando o erro, supondo que o usuário tenha digitado “Algoritmo”, o retorno será:

**Digite uma palavra a ser pesquisada: algoritmo**
**Palavra "algoritmo" não encontrada no texto de origem.**

Simulando o acerto, supondo que o usuário tenha digitado “linguagem”, o retorno será:

**Digite uma palavra a ser pesquisada: linguagem**
**Palavra "linguagem" é uma das palavras do texto de origem**

---

**27 – Escreva uma função que recebe do usuário um número longo qualquer, o convertendo para notação numérica computacional (número original considerado como bytes, convertido de acordo com seu tamanho para kilobytes, megabytes, gigabytes, etc...) e retornando em forma legível ao usuário.**

```
def conv_bytes(bytes):
    sufixo = 'B'
    for i in ['', 'K', 'M', 'G', 'T', 'P', 'E', 'Z']:
        if abs(bytes) < 1024.0:
            return '%3.1f%s%s' % (bytes, i, sufixo)
        bytes /= 1024.0
    return '%.1f%s%s' % (bytes, 'Y', sufixo)

num = int(input('Digite um número: '))
numc = conv_bytes(num)

print(numc)
```

*Converter números entre uma grandeza e outra é algo bastante simples, o pulo do gato está em saber realizar a conversão usando de validações que tornam o resultado final legível e interpretável.*

*Direto ao ponto, inicialmente criamos uma função de nome conv_bytes( ) que por sua vez receberá obrigatoriamente um dado / valor como parâmetro.*

*Dentro do corpo desta função, inicialmente é criada uma variável de nome sufixo, que possui como atributo a string 'B'. Isto se faz necessário pois independentemente da grandeza a qual o número fornecido será sempre inserido o sufixo autocompletando a sigla da grandeza. Em outras palavras, aqui estamos inserindo em*

*todo resultado o sufixo para formar siglas como KB, MB, GB, TB, etc...*

*Em seguida é criado um laço de repetição que percorrerá a lista de prefixos, mapeando para os mesmos uma estrutura condicional onde, se o valor absoluto do dado / valor fornecido para bytes pelo usuário for inferior a 1024.0, é retornado o próprio valor em 3 casas decimais, seguido do prefixo e do sufixo conforme seu tamanho.*

*Caso a condição imposta anteriormente seja validada como False, então é retornado o próprio valor fornecido pelo usuário, acrescido de 1 casa decimal, do prefixo e do sufixo.*

*Por fim, é declarada no corpo geral do código uma variável de nome num que recebe do usuário via input( ) um número imediatamente convertido para número de tipo inteiro como atributo.*

*Também é criada uma variável de nome numc que instancia e inicializa a função conv_bytes( ) por sua vez parametrizada com num, para assim retornar o último valor atribuído a num após as devidas conversões.*

*Finalizando, é exibido em tela via função print( ) o conteúdo de numc.*

Supondo que o usuário tenha digitado 2792283236926, o retorno será:

**Digite um número: 2792283236926**
**2.5TB**

---
---

**28 – Escreva um programa que recebe do usuário um intervalo de tempo em anos, retornando a partir desta informação o número de ocorrências de um dia específico (nesse caso, as segundas-feiras) dentro de um dia específico do mês (nesse caso, no dia primeiro de cada mês do intervalo de tempo previamente definido).**

```
from datetime import datetime

segundas = 0
meses = range(1, 13)

ano_inicio = int(input('Digite o ano inicial: '))
ano_fim = int(input('Digite o ano final: '))

intervalo = []
intervalo.append(ano_inicio)
intervalo.append(ano_fim)

for ano in range(intervalo[0], intervalo[1]+1):
    for mes in meses:
        if datetime(ano, mes, 1).weekday() == 0:
            segundas +=1

print(f'Entre {intervalo[0]} e {intervalo[1]} existem {segundas} segundas-
feiras no primeiro dia do mês.')
```

*Aqui temos mais um exercício o qual podemos resolver de forma simplificada graças a uma das bibliotecas, módulos e pacotes que automatizam certos processos em Python, nesse caso, a manipulação de datas.*

*A lógica do exercício é bastante simples, verificar dentro de um intervalo específico de tempo, quantas segundas-feiras coincidem ser o primeiro dia do mês, e isto pode ser descoberto de modo fácil através do método datetime( ) configurado corretamente para realizar esta validação.*

*Partindo para estrutura do código, como de costume, logo de início realizamos as devidas importações dos pacotes os quais faremos uso, nesse caso, o módulo datetime da biblioteca datetime.*

*Em seguida criamos uma variável de nome segundas, inicialmente de valor 0, que será uma variável de controle para contabilizar quantas segundas-feiras forem encontradas.*

*Também declaramos uma variável de nome meses que recebe como atributo o intervalo de tempo correspondente ao número de meses em um ano. Realizamos esse processo via método range( ), logo, para definir 12 meses parametrizamos tal método em justaposição com 1 e 13.*

*Logo após criamos duas novas variáveis de nomes ano_inicio e ano_fim, respectivamente, que via método input( ) recebem do usuário os números que definem o intervalo de tempo personalizado.*

*Na sequência declaramos uma nova variável, dessa vez de nome intervalo, que inicialmente recebe como seu atributo uma lista vazia, em seguida preenchida com os dados de ano_inicio e ano_fim através do método append( ).*

*Criadas todas variáveis as quais necessitamos, podemos partir para a elaboração do mecanismo de validação. Para isso, criamos um laço de repetição for que percorre o intervalo definido anteriormente pelo usuário, através do método range( ) aqui parametrizado com o referencial inicial mapeado na posição de índice 0 da variável intervalo e com o referencial final mapeado na posição de índice 1 da mesma variável acrescido de uma unidade para que o número final seja o gatilho para o encerramento do mapeamento.*

*Supondo que o usuário tenha digitado anteriormente 2000 e 2010, para que o método range( ) reconheça corretamente esse intervalo deverá ser parametrizado com 2000 e 2011, para retornar uma lista composta de [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010].*

*Indentado e aninhado ao laço de repetição anterior inserimos um novo laço de repetição for, agora percorrendo o intervalo definido em meses.*

*Dentro deste laço de repetição criamos uma estrutura condicional que verifica se o retorno do método datetime( ), aqui parametrizado com os valores retornados a cada ciclo de repetição para ano e mês, mais especificamente se o dia da semana for segunda via weekday( ) == 0 é validado como True. Caso seja, a variável segundas é instanciada, tendo seu valor atualizado acrescido em uma unidade.*

*Por fim, exibimos em tela via função print( ) uma mensagem ao usuário retornando o intervalo especificado pelo mesmo e o número de segundas-feiras.*

Não havendo nenhum erro de sintaxe, o retorno será:

**Digite o ano inicial: 2000**
**Digite o ano final: 2010**
**Entre 2000 e 2010 existem 19 segundas-feiras no primeiro dia do mês.**

---

**29 – Crie uma pasta chamada imagens, entre na pasta e retorne seu caminho completo para uma variável. Exiba em tela o caminho completo da pasta imagens.**

```
import os

os.mkdir('Imagens')
os.chdir('Imagens')

caminho = os.getcwd()

print(caminho)
```

*Em Python, sempre que necessitamos interagir com o sistema operacional, podemos realizar tal interação de maneira descomplicada através dos métodos que a biblioteca os nos fornece.*

*Nesta questão em particular, para criar uma pasta, entrar na pasta criada e retornar seu caminho completo, usaremos dos métodos mkdir( ), chdir( ) e getcwd( ), respectivamente.*

*Sendo assim importamos os no início desse bloco de código, na sequência, diretamente no corpo geral do código inicializamos o método os.mkdir( ), por sua vez parametrizado com a string 'Imagens' para assim criar uma pasta de nome imagens. Essa pasta será criada no mesmo local onde o arquivo Python atual está sendo executado, caso fosse o caso de criar uma pasta em outro local, deveríamos fornecer como parâmetro para mkdir( ) o caminho completo até o diretório raiz.*

*Da mesma forma, mas fazendo uso do método chdir( ), parametrizado com 'imagens', estamos simplesmente entrando dentro da pasta imagens criada anteriormente para trabalhar dentro da mesma.*

*Declaramos então uma variável de nome caminho, que instancia e inicializa o método os.getcwd( ) para retornar o caminho completo da pasta atual.*

*Por fim, via print( ), exibimos em tela o conteúdo atribuído a variável caminho.*

Nesse caso, o retorno será:

**C:/Users/Fernando/Python/Exercicios/Imagens**

---

**30 – Crie um mecanismo otimizado de múltipla escolha que recebe do usuário um determinado número em sua forma numérica e retorna para o mesmo o número escrito por extenso:**

```
def conversor(num):
    match num:
        case 0: return 'Zero'
        case 1: return 'Um'
        case 2: return 'Dois'
        case 3: return 'Três'
        case 4: return 'Quatro'
        case 5: return 'Cinco'

numero = int(input('Digite um número de 0 a 5: '))

print (conversor(numero))
```

*Observando atentamente o enunciado da questão podemos ver que o problema proposto poderia ser facilmente resolvido através de alguma estrutura condicional, que receberia do usuário um determinado número o validando e retornando em forma de string. Porém, um mecanismo de múltipla escolha "otimizado" pode ser desenvolvido implementando match case.*

*Desde a versão 3.10 da linguagem Python temos de forma nativa o mecanismo match case, similar ao switch case de outras linguagens, e que diferentemente de uma estrutura condicional, é superior em performance quando realizada a mesma função.*

*Para nosso exemplo, criamos uma função personalizada de nome conversor( ) que obrigatoriamente recebe como parâmetro um número fornecido pelo usuário.*

*Dentro do corpo desta função, inserimos o iterador mach seguido do parâmetro a ser inspecionado, nesse caso, num.*

*Na sequência criamos as devidas proposições lógicas, neste exemplo em particular, conforme solicitado pela questão, 6 opções pelas quais a entrada fornecida pelo usuário irá aderir. A nível de sintaxe, inserimos o marcador case seguido da proposição e de seu retorno.*

*Em seguida é declarada uma variável de nome numero que através da função input( ) pede que o usuário digite um número dentro do intervalo pré-determinado, validando na atribuição tal número como de tipo inteiro.*

*Por fim exibimos em tela via print( ) o retorno da função conversor( ) por sua vez parametrizada com o dado / valor atribuído para variável numero.*

Não havendo nenhum erro de sintaxe o retorno será:

**Digite um número de 0 a 5: 4**
**‘Quatro’**

---

**31 – Crie uma simples função que soma dois números fornecidos pelo usuário, retornando o resultado de tal operação matemática. Para a função soma( ), crie uma documentação acessível pelo usuário diretamente via código, orientando o mesmo a respeito do funcionamento da função, simulando que tal função é modularizada ou que de alguma forma seu código não é explícito.**

```
# módulo soma.py

def soma(n1, n2):
  ''' Esta é uma simples função de soma de dois números,
  ao chamar a função, dois números devem ser obrigatoriamente
  repassados para a função, a mesma irá retornar o valor
  da soma destes números, em formato float.'''

  return n1 + n2
```

```
from soma import soma

num1 = float(input('Digite o primeiro número: '))
num2 = float(input('Digite o segundo número: '))
soma_numeros = soma(num1, num2)

print(f'O resultado da soma entre {num1} e {num2} é: {soma_numeros}')

print(f'Documentação do módulo soma: \n {soma.__doc__}')
```

*Uma das características comuns à programação em Python é que, uma vez que a mesma é uma linguagem interpretada, conseguimos realizar muita coisa em tempo real, desde buscar*

*informações sobre objetos declarados ao longo do código até consultar documentação “inline”.*

*Assim sendo, desconsiderando outras estruturas de dados e falando apenas de funções, quando criamos nossas funções personalizadas podemos gerar sua documentação inline em forma de Docstrings.*

*Criar uma Docstring é um processo extremamente simples pois tal estrutura é apresentada simplesmente como um comentário longo inserido logo na primeira linha do corpo de uma função, sendo instanciável sempre que o usuário ao chamar a função fazer referência ao método interno __doc__ que retornará o conteúdo da Docstring, caso houver.*

*Posto isto, buscando resolver a questão, vamos separar a resolução em dois momentos, um criando a função com sua respectiva Docstring, outro testando tal funcionalidade.*

*Sendo assim, logo de início criamos uma função de nome soma( ) que receberá obrigatoriamente dois parâmetros n1 e n2 em justaposição.*

*Logo na primeira linha de seu corpo / escopo, inserimos usando a notação de comentário longo um texto explicando didaticamente o funcionamento da função soma( ).*

*Logo após criamos o próprio retorno da função, que nesse caso simplesmente retorna o resultado da soma entre os valores repassados para n1 e n2.*

*Encerrando essa etapa salvamos este arquivo com o nome soma.py.*

*Dando início a segunda parte da resolução da questão, logo na primeira linha de um novo arquivo realizamos a importação do método soma( ) de nosso módulo / pacote soma.*

*Na sequência criamos duas variáveis de nomes num1 e num2, respectivamente, que através do método input( ) recebem como atributo um número digitado pelo usuário.*

*Também declaramos uma terceira variável de nome soma_numeros, que instancia e inicializa nossa função soma( ), parametrizando a mesma com os dados / valores anteriormente repassados para as variáveis num1 e num2.*

*Por fim, exibimos em tela o resultado da operação realizada através da função print( ).*

*Finalizando, exibimos em tela a Docstring da função soma parametrizando nossa função print( ) com nossa função soma, aplicando sobre a mesma o método __doc__.*

Supondo que o usuário tenha digitado 5 e 9, o retorno será:

**O resultado da soma é: 14.0**
**Documentação do módulo soma:**
**Esta é uma simples função de soma de dois números, ao chamar a função, dois números devem ser obrigatoriamente repassados para a função, a mesma irá retornar o valor da soma destes números, em formato float.**

```
help(soma)
```

*Outra forma de consultar a Docstring de uma função é repassar a mesma como parâmetro para a função help.*

Nesse caso, o retorno será:

**Help on function soma in module __main__:**

**soma(n1, n2)**
**Esta é uma simples função de soma de dois números,**
**ao chamar a função, dois números devem ser obrigatoriamente repassados para a função, a mesma irá retornar o valor da soma destes números, em formato float.**

---

**32 – Implemente um sistema de carrinho com inserção e remoção de elementos via comando, da forma mais reduzida possível, porém sem fazer uso de compreensão de listas ou funções vazias, apenas estruturas condicionais simples. O sistema deve se manter ativo até que o usuário digite um comando para sair.**

```
carrinho = []

print('Digite o nome e pressione ENTER para inserir no carrinho.')
print('Digite CORRIGE para remover o último item inserido.')
print('Digite SAIR para fechar o carrinho de compras.')

while (obj := input('Digite um objeto: ')) != 'sair':
    carrinho.append(obj); print(carrinho)
    if obj == 'corrige':
        del(carrinho[-2]); del(carrinho[-1]); print(carrinho)

print(f'Lista de compras final:\n{carrinho}')
```

*Aqui temos um exercício totalmente voltado a explorar nossos conhecimentos no que diz respeito a estruturas de dados não tão rotineiras, pois parece uma incógnita falar em método reduzido sem fazer uso de list ou dict comprehension, assim como a impossibilidade de estruturas condicionais compostas.*

*Porém, analisando um pouco mais friamente a questão, podemos usar de notação reduzida para instruções simples, executando mais de uma por linha de código, assim como podemos manipular a interação com o usuário e com os dados inseridos pelo mesmo através do operador Walrus, que nos permite declarar, instanciar, utilizar e atualizar uma variável diretamente dentro de*

*uma função ou expressão lógica / condicional. Dessa forma será possível contornar as limitações impostas pela questão.*

*Partindo para o código, de início simplesmente declaramos uma variável de nome carrinho, que inicialmente recebe como atributo uma lista vazia a ser atualizada posteriormente.*

*Em seguida criamos as instruções a serem repassadas ao usuário através de textos exibidos em tela via função print( ).*

*Eis que chegamos na parte de interesse do código, onde o mesmo deverá ser reduzido e possuir todas as funcionalidades do mecanismo de carrinho proposto no enunciado da questão.*

*Inicialmente criamos um laço de repetição while, para que toda estrutura de código indentada para a mesma permaneça ativa até que um gatilho seja acionado para encerramento do processo. Assim já alinhamos um dos requisitos do problema proposto.*

*Na própria estrutura while criamos uma condicional simples, definindo que enquanto a expressão for validada como True o ciclo de repetição se repete. Nesse caso, a expressão imposta em si é declarar uma variável de nome obj, que fazendo uso do operador Walrus instancia e inicializa o método input( ), interagindo com o usuário para que o mesmo digite o nome de um item, o retorno dessa função, atribuído a variável obj, é validada como diferente de 'sair' pela condicional.*

*Sempre que o desfecho for True, a variável carrinho é instanciada, adicionando como seu elemento o último dado / valor atribuído para a variável obj. Usando notação reduzida, na mesma linha executamos o comando para uma função print( ) exibir em tela o conteúdo da variável carrinho.*

*Aqui temos o mecanismo que mantém a interação com o usuário ativa, permitindo que o mesmo insira novos itens no carrinho irrestritamente. Próximo passo, criar o mecanismo de exclusão de um item inserido erroneamente.*

*Para isso, criamos outra estrutura condicional simples que simplesmente verifica se o último dado / valor atribuído a variável*

*obj é igual a ‘corrige’, caso tal condição seja validada como True, então usando de notação reduzida removemos via del( ) o próprio elemento ‘corrige’ digitado pelo usuário e inserido no carrinho, assim como o último elemento imediatamente anterior ao comando ‘corrige’. Por fim, ainda na mesma linha damos o comando para que novamente a função print( ) seja executada exibindo o conteúdo de carrinho.*

*Note que isto só é possível pois, diferentemente de uma variável temporária (normalmente **i** para um laço de repetição for), graças ao operador Walrus a variável obj imediatamente após declarada já é instanciável. Em nosso exemplo, declarada diretamente na expressão condicional e instanciada um um novo bloco de código posterior, a variável obj possui o comportamento de uma variável comum, como se houvesse sido declarada no corpo / escopo global do código.*

*Por fim, geramos um último retorno ao usuário, agora diretamente ligado ao desfecho quando a estrutura de repetição while é encerrada.*

Executando o código, podemos realizar alguns testes:

**Digite o nome do item e pressiona ENTER para inserir no carrinho.**
**Digite CORRIGE para remover o último item inserido.**
**Digite SAIR para fechar o carrinho de compras.**
**Digite um objeto: Arroz**
**['Arroz']**
**Digite um objeto: Feijão**
**['Arroz', 'Feijão']**
**Digite um objeto: Carne**
**['Arroz', 'Feijão', 'Carne']**
**Digite um objeto: Macarrão**
**['Arroz', 'Feijão', 'Carne', 'Macarrão']**
**Digite um objeto: corrige**
**['Arroz', 'Feijão', 'Carne', 'Macarrão', 'corrige']**
**['Arroz', 'Feijão', 'Carne']**
**Digite um objeto: Pão**

**['Arroz', 'Feijão', 'Carne', 'Pão']**
**Digite um objeto: sair**
**Lista de compras final:**
**['Arroz', 'Feijão', 'Carne', 'Pão']**

```
while (obj := input('Digite um objeto: ')) != 'sair': carrinho.append(obj); print(carrinho)
    if obj == 'corrige': del(carrinho[-2]); del(carrinho[-1]); print(carrinho)
```

**Caso fosse necessário, poderíamos forçar uma quebra de linha mantendo a indentação apenas em if para reduzir toda a estrutura de código para duas linhas, sendo obviamente menos legível, porém "mais reduzida o possível".*

---

**33 – Escreva um simples módulo / pacote que inspeciona uma determinada pasta, retornando o nome de todos arquivos de imagem contidos na pasta, caso houver.**

```
import os

ext = ('.jpg', '.jpeg', '.bmp', '.png', '.gif')

arquivos = sorted([i for i in os.listdir() if i.endswith(ext)])

print(arquivos)
```

*Quando estamos falando em modularização em Python, basicamente estamos falando em scripts que podem tanto serem importados para outros códigos quanto executados a partir de uma linha de comando via terminal.*

*Especificamente falando do problema proposto, inspecionar uma pasta pode ser feito de forma fácil através do método listdir( ) do módulo nativo os, assim como inspecionar rapidamente um determinado tipo de arquivo pode ser feito através de sua extensão, logo, podemos usar do método endswith( ) para validar algumas extensões de tipos de arquivos de imagem.*

*Inicialmente importamos a biblioteca os, logo após declaramos uma variável de nome ext, que recebe em forma de tupla uma lista com as principais extensões de arquivos de imagens em forma de string.*

*Na sequência declaramos uma nova variável, dessa vez de nome arquivos que usando de list comprehension cria um laço de repetição for que a cada ciclo de repetição verifica, no diretório atual através de os.listdir( ) se algum dos arquivos de tal pasta termina com alguma das extensões descritas na variável ext, aqui repassada como parâmetro para o método endswith( ), tudo ordenado como elementos de uma lista.*

*Por fim, exibimos em tela o conteúdo de arquivos via função print( ).*

O retorno será uma lista vazia caso não exista nenhum arquivo com alguma das extensões definidas, assim como caso existam arquivos, os mesmos serão retornados ordenados alfabeticamente como elementos de uma lista.

---

**34 – Crie um simples mecanismo que valida a inserção de elementos em um dicionário, verificando se o elemento a ser adicionado ao mesmo já existe no dicionário atual.**

```
clientes = {'Fernando' : '991357259',
            'Alberto' : '981120491'}

novo_cliente = input('Digite o nome do cliente: ')

if novo_cliente in clientes.keys():
  print(f'{novo_cliente} já existe na base de dados')
  novo_cliente = input('Digite o nome do cliente: ')
  if novo_cliente not in clientes.keys():
    print(f'Para inserir {novo_cliente} a base de dados, digite um telefone:')
    clientes.__setitem__(novo_cliente, str(input('Digite o telefone: ')))

print(clientes)
```

*Relembrando o básico sobre dicionários em Python, tal estrutura de dados nada mais é do que um contêiner de dados onde seus "elementos" são itens dispostos em forma de chave -> valor. Dessa forma, sempre que estamos a manipular dados de um dicionário, estamos a instanciar uma determinada chave para que o retorno seja o dado / valor situado em seu campo valor.*

*Para criar um mecanismo de validação de entradas em um dicionário, podemos trabalhar diretamente sobre as chaves do mesmo e, caso tal condição não for validada como True, podemos atualizar o conteúdo de tal dicionário adicionando itens ao mesmo via __setitem__( ), método interno que possui autonomia total para manipulação de dicionários.*

*Partindo para a prática, logo da primeira linha de nosso código criamos uma variável de nome clientes que recebe como atributo*

*um dicionário, inicialmente composto por dois pares de chaves / valores.*

*Em seguida, criamos outra variável, agora de nome novo_cliente, que fazendo uso da função input( ) interage com o usuário pedindo que o mesmo digite o nome de um novo cliente.*

*Criadas as variáveis base, podemos partir para criação do mecanismo de validação, nesse caso, uma estrutura condicional simples para validação dos dados fornecidos pelo usuário.*

*Para isso, através de uma estrutura condicional if impomos a condição de que se o último valor atribuído para a variável novo_cliente constar nas chaves do dicionário atrelado a variável clientes, então é retornada uma mensagem de erro ao usuário, instanciando novamente a variável novo_cliente para que o mesmo forneça um novo dado.*

*Repare que para tal estrutura condicional usamos do operador in interagindo especificamente com as chaves do dicionário através de clientes.keys( ).*

*Caso a condição definida anteriormente não tenha sido validada como True, então programamos um novo desfecho para a estrutura condicional através de uma nova estrutura condicional aninhada a primeira. Nesse caso, é exibido em tela ao usuário uma mensagem informando que o nome digitado é válido para um novo cadastro, solicitando que o usuário agora digite um número de telefone atrelado ao nome fornecido.*

*Por fim, instanciamos a variável clientes, aplicando sobre a mesma o método __setitem__( ), aqui parametrizado em justaposição com os dados fornecidos anteriormente para nome e telefone.*

*Encerrando, exibimos em tela via print( ) o conteúdo da variável clientes.*

Executando o código e verificando suas funcionalidades, o retorno será:

**Digite o nome do cliente: Fernando**
**Fernando já existe na base de dados**
**Digite o nome do cliente: Anna**
**Para inserir Anna a base de dados, digite um telefone:**
**Digite o telefone: 991234556**
**{'Fernando': '991357259', 'Alberto': '981120491', 'Anna': '991234556'}**

**----------**

**35 – Uma vez que temos dois conjuntos numéricos iniciais c1 = (2, 4, 6, 8, 10) e c2 = (2, 4, 6, 8, 10) e um terceiro conjunto nomeado como c3, gerado a partir do conteúdo de c2. Equiparando os três conjuntos, verifique se os mesmos possuem mesmo conteúdo e mesmo identificador de memória.**

```
import collections

c1 = (2, 4, 6, 8, 10)
dq1 = collections.deque(c1)

c2 = (2, 4, 6, 8, 10)
dq2 = collections.deque(c2)

c3 = dq2

print(f'Conjunto 1: {dq1}')
print(f'Endereço de Conjunto 1: {id(dq1)}')

print(f'Conjunto 2: {dq2}')
print(f'Endereço de Conjunto 2: {id(dq2)}')

print(f'Conjunto 3: {c3}')
print(f'Endereço de Conjunto 3: {id(c3)}')

if (dq1 == dq2) and (id(dq1) == id(dq2)):
    print('Conjunto 1 e Conjunto 2 possuem os mesmos elementos e são o mes
mo objeto alocado em memória')
if (dq1 == c3) and (id(dq1) == id(c3)):
    print('Conjunto 1 e Conjunto 3 possuem os mesmos elementos e são o mes
mo objeto alocado em memória')
if (dq2 == c3) and (id(dq2) == id(c3)):
    print('Conjunto 2 e Conjunto 3 possuem os mesmos elementos e são o mes
mo objeto alocado em memória')
```

```
if (dq1 == dq2) and (id(dq1) != id(dq2)):
    print('Conjunto 1 e Conjunto 2 possuem os mesmos elementos, porém são objetos diferentes alocados em memória')
if (dq1 == c3) and (id(dq1) != id(c3)):
    print('Conjunto 1 e Conjunto 3 possuem os mesmos elementos, porém são objetos diferentes alocados em memória')
if (dq2 == c3) and (id(dq2) != id(c3)):
    print('Conjunto 2 e Conjunto 3 possuem os mesmos elementos, porém são objetos diferentes alocados em memória')
```

*Aqui temos outra questão que nos coloca na situação de um problema a ser resolvido, que pode ser resolvido de diversas formas. Para fins didáticos, vamos buscar desenvolver uma solução não necessariamente de melhor performance, mas que apresenta claramente a lógica envolvida na resolução no problema.*

*Para que possamos realizar as devidas comparações, sem necessariamente alterar o comportamento dos dados a serem validados, podemos fazer uso de deque, ferramenta da biblioteca collections para manipulação de dados oriundos de contêineres de dados.*

*Como de costume, logo no início de nosso código importamos as bibliotecas, módulos e pacotes os quais faremos uso, nesse caso, a biblioteca collections.*

*Logo após criamos duas variáveis c1 e c2 que replicam os respectivos conjuntos apresentado no enunciado da questão.*

*Imediatamente após declaradas as variáveis c1 e c2, criamos duas novas variáveis de nomes dq1 e dq1, que instanciam e inicializam o método collections.deque( ), por sua vez parametrizado em cada instância com o conteúdo das variáveis c1 e c2. Assim temos os dados originais em seu formato original, e os novos dados, convertidos via deque.*

*Da mesma forma como feito anteriormente, declaramos uma nova variável de nome c3 que recebe como atributo o conteúdo da variável dq2.*

*Uma vez que temos todas variáveis que necessitamos, podemos avançar um passo exibindo em tela o conteúdo de dq1, dq2 e c3 através da função print( ).*

*Exatamente da mesma forma, via print( ), exibimos em tela o número de endereço alocado em memória de tais variáveis simplesmente repassando como parâmetro para print( ) o método id( ) por sua vez parametrizado com as variáveis em questão.*

*A este ponto, haja visto que temos todas as variáveis e seus respectivos identificadores, podemos partir para as fases de validação. Esse processo, como mencionado anteriormente, pode ser feito de diversas formas, aqui, para fins didáticos, vamos nos ater a estruturas condicionais simples e independentes entre si, apenas validando os dados comparativos entre as variáveis.*

*Como primeira estrutura condicional, verificamos se o conteúdo de dq1 e dq2 são o mesmo e também se o número identificador de dq1 é igual ao de dq2. Caso estas duas proposições sejam validadas como verdadeiras, então é exibido em tela uma mensagem via print( ) informando o usuário sobre essa equivalência de dados.*

*Segunda estrutura condicional, aqui verificamos se o conteúdo de dq1 é igual ao de c3, assim como se o identificador de dq1 é igual ao de c3.*

*O mesmo é feito para a terceira estrutura condicional, verificando as mesmas proposições para dq2 e c3. Dessa forma, abrangemos uma parte das validações onde os dados são de fato equivalentes tanto em conteúdo quanto em identificador.*

*Partindo para outra parte das validações, criamos uma nova estrutura condicional onde se o valor de dq1 e dq2 forem iguais e se o identificador de dq1 for diferente de dq1, então é retornado ao*

*usuário uma mensagem referente a equivalência de dados e disparidade de identificadores.*

*O mesmo processo se repete em uma nova condicional verificando se o conteúdo de dq1 é igual ao de c3, e se o identificador de dq1 é diferente do de c3. Da mesma forma é verificado a equivalência de conteúdo de dq1 e c3 associada a divergência sobre o número de identificação de dq1 e de c3.*

*Desse modo, nada performático porém didático, estamos realizando todas as validações necessárias para que possamos entender a disposição dos dados no que diz respeito a como os mesmos são vistos internamente por parte do interpretador.*

Não havendo nenhum erro de sintaxe, o retorno será:

**Conjunto 1: deque([2, 4, 6, 8, 10])**
**Endereço de Conjunto 1: 140131204475040**
**Conjunto 2: deque([2, 4, 6, 8, 10])**
**Endereço de Conjunto 2: 140131204475600**
**Conjunto 3: deque([2, 4, 6, 8, 10])**
**Endereço de Conjunto 3: 140131204475600**
**Conjunto 2 e Conjunto 3 possuem os mesmos elementos e são o mesmo objeto alocado em memória**
**Conjunto 1 e Conjunto 2 possuem os mesmos elementos, porém são objetos diferentes alocados em memória**
**Conjunto 1 e Conjunto 3 possuem os mesmos elementos, porém são objetos diferentes alocados em memória**

---

**36 – Escreva um programa que procura um certo dado / valor a partir de múltiplos dicionários, conforme dado inserido pelo usuário. Os dicionários em questão devem possuir itens domésticos representados com nome e preço. Retorne ao usuário o item consultado, de qual dicionário foi extraído e seu valor em reais.**

```
from collections import ChainMap

cozinha = {'Fogão': 470,
           'Geladeira': 1100,
           'Forno Micro-ondas': 399}
sala = {'Sofá': 799,
        'Estante': 499}
quarto = {'Cama': 1049,
          'Roupeiro': 899,
          'Criado Mudo': 224}
banheiro = {'Pia': 459,
            'Vaso': 499,
            'Lixeira': 49}

x = ChainMap(cozinha, sala, quarto, banheiro)
y = input('Digite o item a ser buscado: ')

if y in cozinha.keys():
  print(f'Informação extraída a partir do dicionário "cozinha"')
  print(f'O valor de {y} é R${x[y]},00')
elif y in sala.keys():
  print(f'Informação extraída a partir do dicionário "sala"')
  print(f'O valor de {y} é R${x[y]},00')
elif y in quarto.keys():
  print(f'Informação extraída a partir do dicionário "quarto"')
  print(f'O valor de {y} é R${x[y]},00')
elif y in banheiro.keys():
```

```
 print(f'Informação extraída a partir do dicionário "banheiro"')
 print(f'O valor de {y} é R${x[y]},00')
else:
 print(f'O dado solicitado não consta em nenhum dicionário')
```

*Uma vez que temos de consultar um certo dado em dois ou mais contêineres de dados, nesse caso, dicionários, uma forma simplificada de realizar tal feito é unificar tais dicionários através da ferramenta ChainMap, pois assim teremos todos os dados encadeados e centralizados em uma única estrutura de dados, facilitando a consulta e manipulação dos mesmos.*

*Diretamente ao código, na primeira etapa realizamos a importação do pacote ChainMap da biblioteca collections.*

*Na sequência são criados 4 dicionários, associados a suas respectivas variáveis cozinha, sala, quarto e banheiro, todos compostos de dois ou mais itens, apenas para fins de exemplo.*

*Em seguida declaramos uma nova variável de nome x, que instancia e inicializa a ferramenta ChainMap, repassando como parâmetro para a mesma as variáveis cozinha, sala, quarto e banheiro. Aqui estamos encadeando os itens dos dicionários em um novo contêiner de dados mapeado.*

*Também declaramos uma variável de nome y, que por sua vez interage com o usuário via função de entrada input( ).*

*Partindo para as validações, de início criamos uma estrutura condicional simples if que verifica se o dado atribuído a variável y, anteriormente fornecido pelo usuário, consta no campos das chaves do dicionário cozinha, através da aplicação do método keys( ) sobre a variável cozinha.*

*Caso tal condição seja verdadeira, é exibido então em tela via função print( ) uma mensagem ao usuário informando o dicionário de origem. Também através da função print( ), usando de f'strings, incorporamos na mensagem máscaras de substituição por onde*

*referenciamos o dado atribuído a variável y e o valor atribuído a variável x na posição y, para que assim retornemos o nome do objeto seguido de seu preço. Como aqui não se faz necessário realizar outro tipo de validação, simplesmente replicamos a mesma estrutura lógica para os demais dicionários.*

Não havendo nenhum erro de sintaxe, ao executar este código o retorno será:

**Digite o item a ser buscado: Roupeiro**
**Informação extraída a partir do dicionário "quarto"**
**O valor de Roupeiro é R$899,00**

**----------**

**37 – Crie três classes representando três funcionários de uma empresa, descreva como atributo de classe primário o cargo, e como atributos secundários seus respectivos nomes e salários, estes dados devem ser imutáveis. Insira os registros destes funcionários em uma lista e exiba em tela o conteúdo da lista de funcionários.**

```
from collections import namedtuple

funcionario = namedtuple('ID', "Cargo Detalhes")
funcionarios = []

func001 = funcionario('Gerente', {'Nome':'Anna', 'Salário':'R$2750,95'})
func002 = funcionario('Fiscal', {'Nome':'Carlos Santos', 'Salário':'R$2480,00'})
func003 = funcionario('Fiscal', {'Nome':'Carlos Silva', 'Salário':'R$2190,00'})

print(f'{func001}\n{func002}\n{func003}')

funcionarios.append(func001)
funcionarios.append(func002)
funcionarios.append(func003)

print(funcionarios)
```

*De acordo com o enunciado da questão, para criação dos registros dos funcionários devemos os fazer forma que seus dados possam ser lidos mas não alterados. Existem diversas formas de se fazer este processo, sendo um dos modos de maior simplicidade e melhor performance sendo através de tuplas nomeadas.*

*Dos tipos de dados básicos em Python bem sabemos que tuplas são contêineres de dados com a característica de guardar seus dados de forma "imutável". Para realizar algo mais elaborado,*

*gerando hierarquias de atributos para um item a ser inserido em uma tupla, podemos realizar tal processo utilizando de uma namedTuple, pois neste tipo de dados temos uma "super tupla" a qual podemos manipular dados de forma mais eficiente e sem problemas de incompatibilidade ou comportamento dos dados em sua composição.*

*Para isso, inicialmente importamos o módulo namedTuple da biblioteca collections.*

*Em seguida, declaramos uma variável de nome funcionario que por sua vez instancia e inicializa o método namedTuple( ), parametrizando o mesmo em justaposição com duas strings, sendo a primeira de maior posição hierárquica, seguida da segunda que podemos interpretar como uma subclasse da primeira.*

*Também criamos uma segunda variável de nome funcionarios que inicialmente recebe como atributo uma lista vazia.*

*Uma vez que temos a estrutura molde para as classes e objetos a serem criados, podemos partir para criação de cada novo objeto que representa cada funcionário.*

*Sendo assim, é declarada uma variável de nome func001 que instancia a variável funcionario, para a partir da mesma gerar seus dados.*

*Dando sequência alimentamos funcionario( ) em justaposição com o parâmetro a ser repassado como 'ID', seguido de duas outras strings, sendo a primeira referente a 'Cargo' e a segunda, por meio de um dicionário de dados, criando as demais informações para 'Detalhes'.*

*Lembrando que aqui o parâmetro definido como 'ID' poderia ser qualquer outra string pois apenas representa o tipo de hierarquia que estamos implementando. Também é válido lembrar que no lugar do dicionário podemos inserir qualquer tipo de contêiner de dados sem problemas de incompatibilidade, pois tudo é resolvido internamente quando o interpretador lê a tupla nomeada e seus dados.*

*Exatamente o mesmo processo é repetido para criação de outras duas tuplas nomeadas, mas referentes as variáveis func002 e func003.*

*Encerrando esta etapa exibimos em tela o conteúdo de func001, func002 e func003 através de nossa velha conhecida função print( ), aqui fazendo uso de f'strings e incorporando em uma mensagem, em suas devidas máscaras de substituição, o conteúdo das variáveis geradas anteriormente.*

*Por fim, apenas obedecendo o problema proposto na questão, criamos uma lista onde cada um de seus elementos será o conteúdo das classes geradas anteriormente.*

*Para isso, instanciamos a variável funcionarios, adicionando à sua lista via append( ) o conteúdo das variáveis func001, func002 e func003.*

*Finalizando, exibimos em tela via print( ) o conteúdo de funcionarios.*

Executando esse código o retorno será:

```
ID(Cargo='Gerente', Detalhes={'Nome': 'Anna', 'Salário': 'R$2750,95'})
ID(Cargo='Fiscal', Detalhes={'Nome': 'Carlos Santos', 'Salário':
'R$2480,00'})
ID(Cargo='Fiscal', Detalhes={'Nome': 'Carlos Silva', 'Salário':
'R$2190,00'})
[ID(Cargo='Gerente', Detalhes={'Nome': 'Anna', 'Salário': 'R$2750,95'}),
ID(Cargo='Fiscal', Detalhes={'Nome': 'Carlos Santos', 'Salário':
'R$2480,00'}), ID(Cargo='Fiscal', Detalhes={'Nome': 'Carlos Silva',
'Salário': 'R$2190,00'})]
```

Apenas fazendo um pequeno adendo, é interessante salientar que, um objeto criado através de uma namedTuple, internamente é lido como um objeto exclusivo, no sentido de que se consultarmos seu tipo de dado, por exemplo type(func001), o retorno será algo como <class '__main__.ID'>, sendo o "tipo de dado" resolvido apenas como 'ID' pois internamente existem diferentes tipos de dados, cada

um com um comportamento específico ditado pela estrutura lógica de uma namedTuple. Na prática isto é totalmente implícito e por padrão se aplicam os mesmos métodos possíveis como em listas e dicionários.

---

**38 – A partir de uma simples base de dados em uma lista, composta por ['1001', '1002', '1003', '1004', '1005', '1006', None, '1008', '1009'], gere uma nova base de dados a partir desta contendo apenas dados válidos. Retorne ao usuário uma mensagem informando a posição de índice do dado inválido da base de dados original.**

```
base = ['1001', '1002', '1003', '1004', '1005', '1006', None, '1008', '1009']

lista = []
inv = 0

for i in base:
  if i is not None:
    lista.append(i)
  else:
    inv = base.index(None)

print(f'Nova lista: {lista}')
print(f'Existe um dado inválido no índice {inv} da lista original')
```

*Por parte da geração de uma nova lista contendo apenas dados "válidos", como tudo em Python, temos diversas formas de realizar tal processo. O fator que complica a resolução desta questão é mapear e retornar a posição de índice do elemento None, sendo a forma mais direta obter esta informação a partir do retorno do método index( ).*

*A nível de código, inicialmente declaramos uma variável de nome base que recebe como atributo a lista proposta pelo enunciado da questão.*

*Em seguida criamos uma variável de nome lista, que inicialmente recebe como atributo uma lista vazia a ser atualizada*

*posteriormente. Esta será a variável que receberá os dados da lista "corrigida".*

*Também criamos uma variável de controle de nome inv, que aqui recebe como atributo o número 0, a ser atualizado com o número de índice mapeado para o dado inválido. Como neste exemplo temos claramente apenas um elemento "inválido", podemos simplesmente ter um número inteiro atrelado a uma variável de controle, número este a ser atualizado com o valor de posição de índice.*

*Na sequência criamos um laço de repetição for que percorre todos os elementos contidos na variável base, retornando a cada ciclo de repetição um dado / valor para variável temporária i.*

*Indentado a este laço de repetição, a cada repetição iremos realizar uma validação baseada em uma estrutura condicional simples. Como estrutura condicional definimos que se o último dado / valor atribuído para variável temporária i não for None, então tal dado / valor é adicionado ao conteúdo de lista, que nesse caso é instanciada, tendo aplicada sobre si o método append( ).*

*Caso a condição definida anteriormente não seja válida, então é instanciada a variável inv, tendo seu valor atualizado com o número de posição de índice mapeado em base através do método index( ), nesse caso, buscando pelo elemento de tipo None.*

*Dessa forma, separamos os elementos válidos para uma nova lista, assim como da lista original mapeamos o elemento invalido extraindo sua posição de índice para referência.*

*Por fim, via print( ), exibimos em tela a nova lista e conforme solicitado pela questão, também exibimos o número de posição de índice da lista original.*

Rodando este script o retorno será:

**Nova lista: ['1001', '1002', '1003', '1004', '1005', '1006', '1008', '1009']**
**Existe um dado inválido no índice 6 da lista original**

---

**39 – Supondo que temos uma lista composta por ['Terça', 'Quarta', 'Quinta', 'Sexta', 'Sabado', 'Domingo', 'Segunda'], reordene os elementos da lista de modo a obter a sequência [‘Segunda’,**
**'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sabado', 'Domingo'].**

```
from collections import deque

semana = deque(['Terça', 'Quarta', 'Quinta', 'Sexta', 'Sabado', 'Domingo', 'Segunda'])
semana.rotate(1)

print(semana)
```

*Observando a lista original, podemos perceber que a lógica envolvida neste problema se dá por deslocar todos os elementos da lista em uma posição, como em um carrossel, para que seus elementos sejam rearranjados de modo a trazer o último elemento para primeira posição, descolando todos os demais “uma casa” para frente.*

*Logo, haja visto que temos de aplicar uma função que itere com cada elemento da lista (e com todos juntamente), podemos transformar a lista em uma fila através de deque( ), para posteriormente resolver o conflito da questão manipulando os elementos da lista como elementos em rotação de uma fila.*

*Para isso, inicialmente importamos da biblioteca collections o método deque.*

*Na sequência declaramos uma variável de nome semana, que instancia e inicializa o método deque( ), repassando como parâmetro para o mesmo a lista sugerida pelo enunciado da questão.*

*Uma vez que temos os elementos da lista dentro de deque( ), podemos simplesmente aplicar sobre a variável semana o método rotate( ), aqui parametrizado com o valor 1 para que o deslocamento dos elementos da fila sejam feitos de uma em uma unidade.*

*Por fim, exibimos em tela via função print( ) o conteúdo da variável semana.*

Nesse caso o retorno será:

**deque(['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sabado', 'Domingo'])**

---

**40 – Uma vez que temos uma lista contendo como elementos próprios uma série de tuplas compostas por uma categoria e por um tipo de exame de imagem, reconstrua essa base de dados em um dicionário, unificando as categorias de exames e agrupando seus subtipos.**

```
data = [('Raios-X', 'Raios-X'),
        ('Magnetismo', 'Ressonância Magnética'),
        ('Ultrassom', 'Ecografia'),
        ('Medicina Nuclear', 'Cintilografia'),
        ('Raios-X', 'Tomografia Computadorizada'),
        ('Medicina Nuclear', 'PET-CT'),
        ('Raios-X', 'Mamografia'),
        ('Raios-X', 'Densitometria Óssea')]

data_dict = {}
for i, j in data:
  data_dict.setdefault(i, []).append(j)

print(data_dict)
```

*Uma vez que estamos a transcrever um tipo de dado em outro, nesse caso uma lista de tuplas para um dicionário, mapeando e agrupando chaves e valores, um modo simples porém pouco eficiente é a partir de um dicionário vazio, definir chaves e valores padrão a partir dos elementos do contêiner de dados de origem.*

*Partindo para prática, logo na primeira linha declaramos uma variável de nome data, que recebe como atributo uma lista composta por uma série de tuplas, que por sua vez, armazenam dados em formato de pares de strings.*

*Em seguida criamos uma nova variável de nome data_dict, que inicialmente recebe como dado atribuído um dicionário vazio*

*através da notação { } (poderia perfeitamente ser através do método construtor de dicionários dict( ), porém por questões de simplicidade não declaramos explicitamente este ou outro método), dicionário este a ser atualizado posteriormente.*

*Logo após criamos um laço de repetição for, que percorre cada um dos elementos de data, desempacotando seus elementos em justaposição para as variáveis temporárias i e j.*

*Indentado a este laço de repetição, instanciamos a variável data_dict, aplicando sobre a mesma o método setdefalut( ), aqui parametrizado também em justaposição com o último dado / valor atribuído para i, seguido de uma lista vazia.*

*Também é aplicado sobre data_dict, em forma de sufixo o método append( ), por sua vez parametrizado com o último dado / valor lido para j.*

*Dessa forma, na primeira camada de construção do dicionário temos o agrupamento das chaves, e em sua segunda camada, dentro de uma lista temos o agrupamento dos respectivos valores.*

*Finalizando, exibimos em tela o conteúdo de data_dict através da função print( ).*

Não havendo nenhum erro de sintaxe, o retorno será:

**{'Raios-X': ['Raios-X', 'Tomografia Computadorizada', 'Mamografia', 'Densitometria Óssea'], 'Magnetismo': ['Ressonância Magnética'], 'Ultrassom': ['Ecografia'], 'Medicina Nuclear': ['Cintilografia', 'PET-CT']}**

---

**41 – Otimize o código anterior fazendo uso de defaultdict( ). Realize a comparação do tempo de processamento necessário para ambos os métodos.**

```
from collections import defaultdict

data = [('Raios-X', 'Raios-X'),
        ('Magnetismo', 'Ressonância Magnética'),
        ('Ultrassom', 'Ecografia'),
        ('Medicina Nuclear', 'Cintilografia'),
        ('Raios-X', 'Tomografia Computadorizada'),
        ('Medicina Nuclear', 'PET-CT'),
        ('Raios-X', 'Mamografia'),
        ('Raios-X', 'Densitometria Óssea')]

def_dict = defaultdict(list)
for i, j in data:
  def_dict[i].append(j)

print(def_dict)
```

*Python é uma linguagem de programação extremamente versátil, oferecendo diversas formas de se realizar uma determinada ação, seja por suas estruturas de dados básicas, seja por estruturas de código criadas por terceiros em formato de bibliotecas, módulos e pacotes que agregam ao núcleo da linguagem não somente mais funcionalidades como estruturas que visam otimizar métodos já existentes.*

*Conforme proposto pela questão, é possível otimizar nosso código substituindo a estrutura de um dicionário comum por um dicionário padrão. Raciocine que o que faz certas estruturas de código modulares serem (normalmente) mais eficientes que estruturas base se dá por conta de suas especificidades.*

*Tenha em mente que um método construtor de dicionários dict( ) possui uma série de ferramentas e métricas internas que muitas vezes não são utilizadas, apesarem de estar carregadas em memória. Já um método especializado como defaultdict( ) possui uma estrutura interna mais enxuta, contendo apenas o necessário para manipulação de seus dados conforme sua proposta.*

*Reescrevendo nosso código, otimizando o mesmo a partir de um dicionário padrão é algo muito simples e intuitivo, conforme veremos a seguir.*

*Logo de início, realizamos a importação da ferramenta defaultdict da biblioteca collections.*

*Em seguida temos exatamente a mesma estrutura de dados da questão anterior, sendo uma lista de tuplas atribuída a variável base.*

*Eis que chegamos então no ponto crucial desta questão, declarando uma nova variável de nome def_dict que instancia e inicializa o método defaultdict( ), que receberá obrigatoriamente um dado em forma de lista.*

*Na sequência criamos um laço de repetição for que percorre todos os elementos de data, mapeando os mesmos e extraindo deus dados / valores a cada ciclo de repetição para suas respectivas variáveis temporárias i e j.*

*Indentado a este laço de repetição, instanciamos a variável def_dict, na posição de índice i, gerando uma chave, atrelando como valor o último dado / valor mapeado para j através do método append( ).*

*Finalizando, exibimos em tela o conteúdo de def_dict via função print( ).*

Como retorno, obtemos:

**defaultdict(<class 'list'>, {'Raios-X': ['Raios-X', 'Tomografia Computadorizada', 'Mamografia', 'Densitometria Óssea'], 'Magnetismo':**

**['Ressonância Magnética'], 'Ultrassom': ['Ecografia'], 'Medicina Nuclear': ['Cintilografia', 'PET-CT']})**

*Partindo para segunda parte do problema, mensurar o tempo de execução de um determinado bloco de código é bastante simples. Uma das formas de maior simplicidade de implementação é fazer o uso do método time( ), inserido antes e depois do bloco de código a ter seu tempo de execução medido.*

```
import time

start = time.time()

data = [('Raios-X', 'Raios-X'),
        ('Magnetismo', 'Ressonância Magnética'),
        ('Ultrassom', 'Ecografia'),
        ('Medicina Nuclear', 'Cintilografia'),
        ('Raios-X', 'Tomografia Computadorizada'),
        ('Medicina Nuclear', 'PET-CT'),
        ('Raios-X', 'Mamografia'),
        ('Raios-X', 'Densitometria Óssea')]

data_dict = {}
for i, j in data:
  data_dict.setdefault(i, []).append(j)

print(data_dict)

end = time.time()

tempo = end - start

print(tempo)
```

*Na prática, na primeira linha de nosso código importamos o módulo time.*

*Logo na sequência já inserimos uma variável de nome start, que instancia e inicializa o método time.time( ). Aqui internamente o que está sendo feito é simplesmente o mapeamento do horário inicial de processamento do bloco de código a seguir.*

*Como bloco de código em si, apenas implementamos exatamente o mesmo código criado para a questão anterior.*

*Logo após o término deste bloco, é criada uma variável de nome end, que recebe como atributo o retorno de time.time( ), agora referente ao horário de término de leitura e processamento do bloco de código anterior.*

*Finalizando, declaramos uma última variável, dessa vez de nome tempo, que simplesmente recebe como atributo o resultado da subtração do tempo final pelo tempo inicial de processamento.*

*Por fim, via print( ) exibimos em tela o conteúdo da variável tempo.*

Nesse caso, o retorno será:

**{'Raios-X': ['Raios-X', 'Tomografia Computadorizada', 'Mamografia', 'Densitometria Óssea'], 'Magnetismo': ['Ressonância Magnética'], 'Ultrassom': ['Ecografia'], 'Medicina Nuclear': ['Cintilografia', 'PET-CT']}**
**0.0024955997467041016**

```
import time

start = time.time()

from collections import defaultdict

data = [('Raios-X', 'Raios-X'),
        ('Magnetismo', 'Ressonância Magnética'),
```

```
        ('Ultrassom', 'Ecografia'),
        ('Medicina Nuclear', 'Cintilografia'),
        ('Raios-X', 'Tomografia Computadorizada'),
        ('Medicina Nuclear', 'PET-CT'),
        ('Raios-X', 'Mamografia'),
        ('Raios-X', 'Densitometria Óssea')]

def_dict = defaultdict(list)
for i, j in data:
  def_dict[i].append(j)

print(def_dict)

end = time.time()

tempo = end - start

print(tempo)
```

*Exatamente o mesmo processo é realizado para o bloco de código otimizado via defaultdict( ). Para fins de precisão, inserimos inclusive a importação da ferramenta defaultdict( ) para assim mensurar o tempo total, de todas estruturas de dados envolvidas.*

Nesse caso, obtemos como retorno:

**defaultdict(<class 'list'>, {'Raios-X': ['Raios-X', 'Tomografia Computadorizada', 'Mamografia', 'Densitometria Óssea'], 'Magnetismo': ['Ressonância Magnética'], 'Ultrassom': ['Ecografia'], 'Medicina Nuclear': ['Cintilografia', 'PET-CT']})**
**0.0011370182037353516**

*Realizando uma breve comparação podemos ver que de fato, o bloco de código otimizado via defaultdict( ) mesmo com sua faze de importação de ferramentas se mostrou mais eficiente que o bloco*

*de código escrito da maneira comum. Importante salientar que como estamos nestes exemplos a trabalhar com estruturas de dados pequenas, a diferença medida e retornada do tempo de processamento entre os dois blocos de código foi realmente ínfima. Em situações reais, fazendo uso de contêineres de dados maiores, este tempo de processamento poupado será maior e de melhor relevância para a aplicação final.*

---

**42 – Supondo que temos um dicionário, representando usuários de um sistema, onde cada item do dicionário é um usuário descrito por seu id, nome, sobrenome e e-mail, exiba em tela de forma estruturada (exatamente como escrita no corpo do código) o conteúdo desse dicionário.**

```
from pprint import pprint

data = {"usuarios": [{"Id": 1,
                      "Nome": "Rafael",
                      "Sobrenome": "Moraes",
                      "email": "rmoraes@gmail.com"},
                      {"Id": 2,
                      "Nome": "Paula",
                      "Sobrenome": "Santos",
                      "email": "paulass@hotmail.com"},
                      {"Id": 3,
                      "Nome": "Daniel",
                      "Sobrenome": "Pires",
                      "email": "dr.daniel@gmail.com"}]}

pprint(data)

print(data)
```

*Solucionar essa questão é algo bastante simples, desde que saibamos onde buscar as ferramentas certas para tal solução. Nesse caso, como bem sabemos, o exibir em tela o conteúdo de um dicionário através de nossa velha conhecida função print( ) fará com que o conteúdo do mesmo seja apresentado por extenso, de modo que compromete sua legibilidade, principalmente se estamos usando de longas estruturas de dados.*

*Porém, em alternativa a função print( ) nativa do núcleo Python temos a função pprint( ), da biblioteca pprint, que por sua vez tem como característica a exibição em tela / terminal de dados mantendo sua forma original, exatamente o que buscamos para esta questão.*

*Direto ao código, da biblioteca pprint importamos a ferramenta pprint.*

*Na sequência é declarada uma variável de nome data, que recebe como atributo um dicionário composto por alguns itens situados em dicionários dentro de dicionários.*

*Através de pprint( ), por sua vez parametrizado com a variável data, exibimos em tela seu conteúdo.*

Nesse caso o retorno será:

```
{'usuarios': [{'Id': 1,
               'Nome': 'Rafael',
               'Sobrenome': 'Moraes',
               'email': 'rmoraes@gmail.com'},
              {'Id': 2,
               'Nome': 'Paula',
               'Sobrenome': 'Santos',
               'email': 'paulass@hotmail.com'},
              {'Id': 3,
               'Nome': 'Daniel',
               'Sobrenome': 'Pires',
               'email': 'dr.daniel@gmail.com'}]}
```

*Apenas para fins de comparação, exibimos o conteúdo de data em tela através da função print( ).*

Como retorno, obtemos:

```
{'usuarios': [{'Id': 1, 'Nome': 'Rafael', 'Sobrenome': 'Moraes', 'email': 'rmoraes@gmail.com'}, {'Id': 2, 'Nome': 'Paula', 'Sobrenome': 'Santos', 'email': 'paulass@hotmail.com'}, {'Id': 3, 'Nome': 'Daniel', 'Sobrenome': 'Pires', 'email': 'dr.daniel@gmail.com'}]}
```

---

**43 – Crie um mecanismo que gera um número pseudorrandômico, no intervalo entre 0 e 1, exibindo tal número em tela com 5 casas decimais após a vírgula.**

```
import random

random.seed(42)

var = random.random()

print(f'{var:.5}')
```

*Quando estamos falando da geração de números aleatórios em Python, podemos automatizar tal processo através das ferramentas fornecidas pela biblioteca random.*

*A particularidade envolvida neste problema se dá no fato de se gerar um número pseudorrandômico, ou seja, um número supostamente aleatório o qual podemos reproduzir para certos contextos.*

*Lembrando que todo e qualquer número "aleatório" em Python não é totalmente randômico, porém, cada vez gerado tal número o mesmo será um número diferente retornado ao usuário. Para controlarmos a aleatoriedade, no que diz respeito a gerar um número aleatório, porém reproduzível, no sentido de ser possível gerar novamente o mesmo número, devemos atribuir uma seed ao processo. O valor de uma seed será o critério o qual internamente as funções de geração de números usará como base, e através deste mesmo número sempre chegaremos ao mesmo resultado.*

*Sem mais delongas, na primeira linha de nosso código importamos a biblioteca random.*

*Logo em seguida, diretamente no corpo de nosso código instanciamos e inicializamos o método random.seed( ), aqui definido*

*como parâmetro o número 42.*

*Na sequência é criada uma variável de nome var, que por sua vez instancia e inicializa o método random.random( ), que por padrão gera um número aleatório dentro do intervalo entre 0 e 1.*

*Por fim, através da função print( ) exibimos em tela o conteúdo da variável var, aplicando a notação :.5 para que assim sejam exibidos os primeiros 5 números após a vírgula.*

Não havendo nenhum erro de sintaxe, o retorno será:

**0.63943**

---

**44 – Dada a lista ['python', 'java', 'php', 'c++', 'c#', 'javascript', 'kotlin', 'r'], retorne duas novas listas geradas a partir da lista inicial, com seus elementos reordenados de forma aleatória, porém reproduzível.**

```
import random

random.seed(32)
linguagens = ['python', 'java', 'php', 'c++', 'c#', 'javascript', 'kotlin', 'r']
random.shuffle(linguagens)
print(linguagens)

random.seed(42)
random.shuffle(linguagens)
print(linguagens)
```

*Nos mesmos moldes do exercício anterior, para gerar uma nova lista a partir da lista de origem, obedecendo o critério "aleatória porém reproduzível", usaremos de uma seed acompanhada de um método gerador.*

*Sendo assim, na primeira linha de nosso código importamos a biblioteca random, em seguida já definimos uma seed específica através do método random.seed( ).*

*É então criada uma variável de nome linguagens, que recebe como seu atributo uma réplica da lista descrita no enunciado da questão.*

*Diretamente no corpo do código instanciamos e executamos o método random.shuffle( ), por sua vez parametrizado com o conteúdo da variável linguagens. Dessa forma, o conteúdo original da variável linguagens será aleatoriamente redistribuído com base na seed anteriormente gerada.*

*Exibindo em tela o conteúdo de linguagens via função print( ), o esperado é que seja uma versão de linguagens com seus elementos ordenados de forma diferente se comparado com a disposição inicial.*

*Como o exercício nos pede o retorno de duas listas, repetimos o processo gerando uma nova seed, sorteando novamente os elementos de linguagem e exibindo em tela seu conteúdo.*

Nesse caso, o retorno será:

**['c#', 'python', 'c++', 'javascript', 'php', 'kotlin', 'r', 'java']**
**['javascript', 'php', 'r', 'java', 'c++', 'kotlin', 'c#', 'python']**

---

**45 – Supondo que a lista data = [59.19, 72.05, 66.82, 81.15, 66.33, 94.87, 99.65, 70.13, 55.75, 87.71 , 95.43, 93.17, 98.54, 68.31, 59.24, 88.94, 79.44, 83.91, 84.4, 68.21] é o retorno dos últimos 20 registros de um sensor qualquer, retorne a média simples, a média harmônica e a média geométrica destes valores. Os resultados devem conter a precisão de 5 casas decimais.**

```
import statistics
import numpy as np

data = [59.19, 72.05, 66.82, 81.15, 66.33, 94.87, 99.65, 70.13, 55.75, 87.71,95
.43, 93.17, 98.54, 68.31, 59.24, 88.94, 79.44, 83.91, 84.4, 68.21]

print(f'Média Simples: {round(statistics.mean(data), 5)}')
print(f'Média Harmônica: {round(statistics.harmonic_mean(data), 5)}')

data = np.array(data)
media_geom = round(data.prod()**(1.0/len(data)), 5)

print(f'Média Geométrica: {media_geom}')
```

*No âmbito da análise de dados, uma prática comum é a partir de alguns dados de entradas, sejam estes bases de dados de qualquer tipo / estrutura, aplicar algumas métricas para produzir novos dados a partir dos de origem.*

*Em outras palavras, uma vez que temos uma lista composta por uma série de valores numéricos, podemos retornar a partir dos mesmos dados como valor médio, média harmônica, média*

*geométrica, produto escalar, etc... E nesse processo devemos ter capacidade de abstrair a "fórmula" para que se calculem tais dados.*

*Logicamente, muita coisa pode ser feita de forma automatizada e otimizada através de funções de bibliotecas já existentes, porém nem sempre encontraremos todas as soluções desta forma, logo, para questões muito específicas pode ser necessário a implementação manual de uma fórmula / equação matemática.*

*Partindo para a resolução do exercício, nas duas primeiras linhas de nosso código importamos as bibliotecas statistics e numpy, pois através das mesmas automatizaremos alguns processos.*

*Logo após declaramos uma variável de nome data, que por sua vez recebe como atributo, em forma de lista de acordo com a notação de colchetes, a lista do enunciado da questão.*

*Na sequência, diretamente via função print( ), utilizando de f'strings, exibimos em tela o retorno referente a média simples, através do método statistics.mean( ) aqui parametrizado com o conteúdo da variável data. Também atendendo as particularidades propostas pela questão, aplicando o método round( ), repassando como parâmetros em justaposição para o mesmo o retorno do método mean( ), seguido do número 5 para que o resultando, se possível, tenha 5 casas decimais.*

*Para a média harmônica repetimos praticamente o mesmo procedimento, porém fazendo uso do método statistics.harmonic_mean( ).*

*Eis que então nos deparamos com um real desafio. Como calcular a média geométrica dos valores de nossa base de dados se a princípio não temos uma função pré-moldada para esta finalidade?*

*Lembrando que conceitualmente médias são calculadas sobre grupos de dados / valores, e que especificamente uma média geométrica é calculada sobre os valores medianos dentro de um certo grupo de limite bem definido.*

$$M = \sqrt[n]{d_1 . d_2 . \ldots . d_n}$$

*Através de sua fórmula padrão, podemos observar que a média geométrica é calculada como a raiz aplicada sobre um conjunto de elementos multiplicados entre si. Pela literatura, segundo os autores da área, a regra que se aplica é a raiz enésima do produto dos elementos, sendo o valor da raiz definido de acordo com o número de elementos. Por exemplo, a média geométrica de um grupo de 3 valores servirá como parâmetro para que seja aplicada uma raiz cúbica sobre tais elementos.*

*Retomando nossa linha de raciocínio, vamos buscar aplicar esta fórmula manualmente para assim produzir o resultado esperado.*

*Para isso, instanciamos a variável data, a convertendo para uma array do tipo numpy através do método np.array( ). Este processo visa converter a estrutura de dados de lista simples para uma array que nos trará, além de uma melhor performance de processamento, um leque maior de possibilidades.*

*Na sequência criamos uma nova variável de nome media_geom que recebe como atributo o retorno da expressão data.prod( ) ** 1.0/len(data). Em outras palavras, na primeira parte da equação estamos gerando o produto comum dos dados de data, elevando este valor ao resultado da divisão de 1 pelo número de elementos presentes em data, tudo isso novamente com valor declarado com 5 casas decimais.*

*Por fim, podemos através de funções print( ) exibir em tela os resultados gerados para a média simples, harmônica e geométrica dos dados.*

Nesse caso, o retorno será:

**Média Simples: 78.662**
**Média Harmônica: 76.205**
**Média Geométrica: 77.44087**

---

**46 – Supondo que para um determinado propósito temos uma classe de nome Menu, que quando instanciada recebe um dado / valor que deve obrigatoriamente ser de tipo numérico para criação de seu respectivo objeto / atributo de classe. A validação por sua vez deve ocorrer a partir do método construtor da classe.**

```
import numbers

class Menu:
  def __init__(self, n):
    if isinstance(n, numbers.Number) and n > 0:
      self.n = n
    else:
      raise TypeError('Valor deve ser numérico e maior que zero')

maquina1 = Menu(5)

maquina2 = Menu('cinco')
```

De acordo com o enunciado da questão, temos duas validações a serem realizadas, sendo que somente a partir das mesmas o objeto / atributo de classe ser gerado.

Para isso, inicialmente importamos a biblioteca numbers, que nos será útil para a validação a ser realizada verificando se o dado / valor inserido é do tipo numérico.

Logo após criamos a classe Menu, indetado ao corpo da mesma criamos seu método construtor __init__( ), inserindo em seus campos por justaposição a própria instância da classe, seguido de um objeto n.

Indentado ao corpo do próprio método construtor da classe inserimos uma estrutura condicional simples, que inicialmente verifica se o dado / valor repassado é numérico através do método isinstance( ), por sua vez parametrizado de n e do método numbers.Number, aqui apenas carregado e não inicializado, apenas para retornar que o dado é numérico ou não. Após o operador and inserimos outra condicional simples, validando simplesmente se o valor de n é maior que 0.

Lembrando que através do operador and, apenas se as duas proposições na expressão declarada para a estrutura condicional forem validadas como True é que o desfecho será executado, caso contrário, via raise, aqui levantando um erro de tipo de dado, exibe ao usuário uma mensagem de erro.

Para teste de funcionalidade de nossa classe Menu, vamos fazer uso da mesma através de dois objetos, sendo o primeiro fazendo o uso correto da classe e um segundo objeto simulando o erro.

Para isso, de volta ao corpo geral de nosso código, são declaradas duas variáveis de nomes maquina1 e maquina2, respectivamente, que instanciam a classe Menu, repassando para a mesma algum dado / valor para alimentar seu atributo n.

Executando o código é esperado que para maquina2 seja levantada uma exceção, pois ‘cinco’, escrita em forma de string não terá como ser validada de acordo com os critérios estabelecidos.

O retorno será:

```
---------------------------------------------------------------------------
TypeError   Traceback (most recent call last)
<ipython-input-14-5e24b50c85e6> in <module>()
    10 maquina1 = Menu(5)
    11
---> 12 maquina2 = Menu('cinco')
```

<ipython-input-14-5e24b50c85e6> in __init__(self, n)
      6       self.n = n
      7     else:
----> 8       raise TypeError('Valor deve ser numérico e maior que zero')
      9
     10 maquina1 = Menu(5)

TypeError: Valor deve ser numérico e maior que zero

---
---

**47 – Como bem sabemos, operações matemáticas em Python realizadas a partir de valores numéricos muito próximos tendem a gerar números com casas decimais, valores em casas decimais normalmente "arredondados" por questões de performance. Porém, em computação científica pode ser necessário números com grande número de casas decimais pois estes valores, quanto mais extensos, simbolizam maior precisão. Realize a simples soma de dois valores 0.1 e 0.2, retornando um terceiro número de no mínimo 20 casas decimais.**

```
import decimal as d

resultado = d.Decimal(0.1) + d.Decimal(0.2)

print(resultado)
```

*Concordando com o enunciado da questão, de fato as operações matemáticas simples, fazendo uso das funções padrão da linguagem ou diretamente sobre seus operadores aritméticos, tendem a produzir resultados "arredondados".*

*Para atender a especificidade da questão, mínimo de 20 casas decimais, teremos de fazer uso de funções voltadas a este tipo de precisão.*

*Para isso, logo na primeira linha de nosso código importamos a biblioteca decimal, a referenciando como d apelas por convenção.*

*Em seguida é criada uma variável de nome resultado, que recebe como atributo o resultado da soma de 0.1 e 0.2, porém, fazendo o uso do método d.Decimal( ), que internamente lê um*

*determinado número em toda sua complexidade. Para que tenhamos um resultado de soma preciso este passo se faz necessário para que cada elemento incluso na soma também tenha valores precisos a serem considerados.*

*Por fim, exibimos em tela o resultado da soma atribuído a variável resultado, usada como parâmetro para função print( ).*

*Por padrão, um número carregado pela ferramenta Decimal possui um número de 28 casas decimais, logo, o resultado esperado tende a ser próximo a esse nível de precisão.*

Executando o código, o retorno será:

**0.3000000000000000166533453694**

---
---

**48 – Uma vez que temos as frações 1/4 e 5/8, realize a soma dessas frações, exibindo em tela detalhadamente seus numeradores, denominadores e o próprio resultado da soma.**

```
from fractions import Fraction

frac1 = Fraction(1, 4)
print(f'Fração: {frac1}')
print(f'Numerador: {frac1.numerator}')
print(f'Denominador: {frac1.denominator}')

frac2 = Fraction(5, 8)
print(f'Fração: {frac2}')
print(f'Numerador: {frac2.numerator}')
print(f'Denominador: {frac2.denominator}')

frac3 = Fraction(frac1) + Fraction(frac2)
print(f'Fração: {frac3}')
print(f'Numerador: {frac3.numerator}')
print(f'Denominador: {frac3.denominator}')

print(f'Soma das Frações: {frac3}')
```

*Trabalhar com números fracionados pode ser um tanto quanto abstrato, principalmente quando temos que os codificar e não necessariamente temos um símbolo / marcador / operador dedicado a tal propósito. Na notação convencional costumamos escrever números fracionados utilizando de uma barra / para sua separação, porém em Python, e não somente em Python, este símbolo costuma ser um operador de divisão simples. Logicamente, em uma fração estamos dividindo seu numerador pelo seu respectivo denominador, porém, para fins de documentação ou escrita via código realizar a*

*simples representação visual de uma fração pode ser uma dor de cabeça dependendo o contexto.*

*Para realizar corretamente as operações entre números fracionários, apenas para fins de exemplo, usaremos da ferramenta Fraction, da biblioteca fractions.*

*Sendo assim, logo no início de nosso código, da biblioteca fractions importamos a ferramenta Fraction através dos comandos from e import.*

*Em seguida criamos uma variável de nome fract1, que instancia e inicializa a ferramenta Fraction( ), parametrizando a mesma com os números 1 e 4, que aqui por justaposição serão identificados como numerador e denominador.*

*Através da função print( ), fazendo uso de f'strings e suas máscaras de substituição, exibimos em tela o conteúdo da variável fract1.*

*Nos mesmos moldes, parametrizando print( ) com frac1.numerator e frac1.denominator exibimos em tela estes dados separadamente.*

*Exatamente o mesmo procedimento é realizado para a segunda fração proposta pelo enunciado da questão, tudo atribuído a uma nova variável, agora de nome frac2.*

*É então declarada uma terceira variável, dessa vez nomeada como frac3, que recebe a soma das frações definidas para frac1 e frac2. O processo aqui, evitando um comportamento indesejado dos números, também é fazendo uso de Fraction( ), por sua vez parametrizado com frac1 e frac2, respectivamente, obtendo sua soma através do operador comum de soma +.*

*Encerrando, exibimos em tela o resultado da soma das frações repassando frac3 como parâmetro para função print( ).*

Nesse caso, o retorno será:

**Fração: 1/4**
**Numerador: 1**

**Denominador: 4**
**Fração: 5/8**
**Numerador: 5**
**Denominador: 8**
**Fração: 7/8**
**Numerador: 7**
**Denominador: 8**
**Soma das Frações: 7/8**

**----------**

**49 – Crie um mecanismo de desempacotamento de dados, capaz de reduzir listas aninhadas para uma só lista comum sem alterar o comportamento de seus elementos. A lista a ser usada de referência é: [[1, 2, [3, 4, 5, 6, 7, 8]], [9, 10, 11, 12]].**

```
from pydash import py_

lista1 = [[1, 2, [3, 4, 5, 6, 7, 8]], [9, 10, 11, 12]]

lista2 = py_.flatten_deep(lista1)

print(lista2)
```

*Quando estamos falando de manipulação de dados em contêineres de dados, assim como praticamente tudo em Python, temos uma vasta gama de possibilidades para se chegar ao mesmo resultado final, cabendo ao desenvolvedor escolher o método o qual julga melhor para tal propósito.*

*No caso de listas dentro de listas, em primeiro lugar devemos nos lembrar que cada elemento de uma lista é um objeto mapeado e indexado de acordo com sua lista de origem, logo, quando temos uma lista aninhada a outra, temos "camadas" de indexação as quais sofrerão alterações de acordo com o modo ao qual estamos manipulando a lista principal.*

*Uma das formas bastante simples, no que diz respeito ao desempacotamento de listas sobre listas, é usar da ferramenta py_ da biblioteca pydash.*

*A nível de código, inicialmente realizamos a importação do módulo py_ da biblioteca pydash através dos comandos from e import.*

*Em seguida declaramos uma variável de nome lista1, que por sua vez recebe como atributo a lista proposta pela questão. Em tal lista, podemos ver que seu elemento de índice 0 é uma lista onde seu elemento de índice número 2 é outra lista... logo, teremos que reduzir estas camadas para correta geração da lista solicitada pelo enunciado da questão.*

*Para isso, criamos uma nova variável de nome lista2, que instancia e inicializa o método py_.flatten_deep( ), por sua vez parametrizado com nossa variável lista1. Internamente o que esse método fará é o desempacotamento de todos os elementos das sublistas para uma lista principal, uma matriz unidimensional contendo todos elementos. Apenas realizando um pequeno adendo, para casos onde temos apenas uma lista aninhada a outra, o processo pode ser feito através do método py_.flatten( ), e em casos onde temos múltiplas camadas de listas aninhadas, se faz necessário o uso do método py_.flatten_deep( ).*

*Por fim, exibimos em tela o conteúdo de lista2 através de nossa velha conhecida função print( ).*

O retorno será:

**[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]**

**----------**

**50 – Implemente dois mecanismos de extração de dados a partir de dicionários aninhados, sendo ao menos um deles o método convencional, utilizando da notação padrão de extração de dados a partir das chaves de um dicionário.**

```
item = {'TV':{'Mostruário':{'LG':'14p',
                            'Sony':'29p',
                            'Philips':'32p'},
              'Estoque':{'Sony':'29p',
                         'LG':'32p'}}}

print(item['TV']['Estoque']['LG']) # Método convencional

print(py_.get(item, 'TV.Estoque.LG')) # Método otimizado
```

*Eis que temos em mãos mais uma vez um exercício para aguçar nossa curiosidade a respeito de métodos alternativos aos métodos padrão, de modo a aprender sobre formas reduzidas e / ou otimizadas de se realizar um determinado processo.*

*Em nosso exemplo, temos uma variável declarada de nome item, que recebe como atributo um dicionário que de imediato podemos identificar como uma estrutura de dados composta por dicionários dentro de dicionários.*

*Para instanciarmos objetos inseridos em camadas "dentro de camadas", devemos usar da notação de fatiamento múltiplo, onde cada par de colchetes [ ] representa uma camada abaixo da principal. Em outras palavras, para buscar uma determinada informação de um objeto situado em um dicionário dentro de outro dicionário, é necessário mostrar ao interpretador do núcleo Python as camadas de dados até o objeto.*

*Na prática, diretamente como parâmetro de nossa função print( ) instanciamos o dicionário atribuído a variável item, na posição de índice 'TV' da primeira camada, posição de índice 'Estoque' da segunda camada, posição de índice 'LG' na camada final, onde está o objeto em questão. Assim temos o método convencional, de legibilidade simples, porém pouco eficiente.*

*Partindo para o método alternativo e, a princípio, otimizado, podemos fazer uso do método get( ) da biblioteca pydash, utilizada no exercício anterior.*

*Novamente diretamente como parâmetro da função print( ), agora instanciamos e inicializamos o método py_.get( ), que recebe em justaposição dois parâmetros, sendo o primeiro o dicionário de origem, e o segundo, em forma de string, a notação de subdiretório até o objeto em questão, nesse caso, 'TV.Estoque.LG', representando algo como \TV\Estoque\LG.*

Não havendo nenhum erro de sintaxe, o retorno será:

**32p**
**32p**

```
item = {'TV':{'Mostruário':{'LG':'14p',
                            'Sony':'29p',
                            'Philips':'32p'},
              'Estoque':{'Sony':'29p',
                         'LG':['32p',
                               '55p']}}}

print(py_.get(item, 'TV.Estoque.LG'))
print(py_.get(item, 'TV.Estoque.LG[0]'))
print(py_.get(item, 'TV.Estoque.LG[1]'))
```

*Apenas realizando um pequeno adendo, caso dentro do mesmo contêiner de dados existam valores para as chaves dos dicionários compostas por múltiplos elementos, podemos combinar a notação padrão, de fatiamento por conta da posição de índice, com a notação empregada pela biblioteca pydash.*

*No exemplo acima, para a chave LG do dicionário Estoque inserimos dois elementos representando dois tamanhos diferentes em polegadas.*

*Nos mesmos moldes como feito para o exemplo anterior, ao repassar como parâmetro para py_.gey( ) 'TV.Estoque.LG' teremos o retorno de todos os elementos da lista associados a este objeto, usando da notação de fatiamento [0] retornaremos apenas o elemento situado na primeira posição de índice, da mesma forma pelo fatiamento [1] teremos os dados referentes ao segundo elemento da lista-valor atribuída a chave 'LG'.*

Nesse caso, o retorno será:

**['32p', '55p']**
**32p**
**55p**

**----------**

# Considerações Finais

Como costumo escrever no fechamento de meus livros... E assim concluímos esse pequeno tratado, dessa vez de exercícios resolvidos e comentados em Python.

Espero que esses exercícios e seus respectivos comentários / explicações tenham de fato lhe ajudado a entender os conceitos mais comumente usados quando estamos programando em Python.

Caso você já possua alguma bagagem de conhecimento sobre os tópicos abordados, espero que tais exemplos tenham ao menos lhe ajudado a ter uma ótica diferente sobre os problemas propostos, de modo que agregasse a seu conhecimento diferentes modos de se solucionar certos problemas.

Por fim, resta meu sincero agradecimento por ter adquirido esta humilde obra e que a leitura da mesma tenha sido tão prazerosa a você quanto foi a mim por escrevê-la.

Um forte abraço.

Fernando Feltrin