Lista 8 - Dicionários

Submissão: Crie um programa para cada questão abaixo, ou seja um executável como mostrado em aula, depois compacte eles em um arquivo .zip(só aceitarei essa forma de compactação) e submeta ao sigaa.

Exemplo:

0. Crie um programa que define 4 funções representando as operações básicas da aritmética: add, sub, mul e div. Cada função tem 2 parâmetros inteiros. O programa deve pedir ao usuário o tipo de operação (1 p/ adição, 2 p/ subtração, 3 p/ multiplicação e 4 p/ divisão inteira) e os valores dos argumentos. Imprimirá ao final o resultado da operação.

Entrada	Saída
1 2 3	5
2 2 3	-1
3 2 3	6
4 2 3	0

Exemplo do código fonte:

```
def add(num1, num2):
    return num1 + num2
def sub(num1, num2):
    return num1 - num2
def mul(num1, num2):
    return num1 * num2
def div(num1, num2):
    return num1//num2
```

```
if __name__ == “__main__”:
	oper = int(input())
	num1 = int(input())
	num2 = int(input())
	resultado = 0

	if(oper == 1):
		resultado = add(num1, num2)
	elif(oper == 2):
		resultado = sub(num1, num2)
	elif(oper == 3):
		resultado = mul(num1, num2)
	else:
		resultado = div(num1, num2)

	print(resultado)
```

O aluno 000000 criou o código fonte acima no arquivo 000000_q0.py. Nas próximas questões ele fez algo semelhante: 000000_q1.py, 000000_q2.py, 000000_q3.py. Após, as questões serem respondidas ele compactou usando .zip. Não é .rar, .7zip ou etc. É .zip. Por quê tem que ser .zip? Porque o .rar, usado por alguns, não funciona em qualquer sistema operacional.

1. (0,05) Escreva uma função que, dada uma string, faça um dicionário vocabulário que armazena o número de vezes que uma palavra aparece, faça o mesmo para cada letra. Não precisa estar em ordem alfabética. A função deve ter a assinatura a seguir:

def constroi_vocabulario(frase: str) -> NoneType:

Ou seja, ela receberá a string, construirá dois dicionários e imprimirá os dois.

Um exemplo de execução da função está definido na tabela abaixo.

Entrada	Saída

constroi_vocabulario(“isso e um teste az teste isso”)	Palavras:
	{
	'isso': 2,
	'e': 1,
	'um': 1,
	'teste': 2,
	'az': 1
	}
	Letras:
	{
	'i': 2,
	's': 6,
	'o': 2,
	' ': 6,
	'e': 5,
	'u': 1,
	'm': 1,
	't': 4,
	'a': 1,
	'z': 1
	}

2. Um vetor esparso é um vetor cujas entradas são quase todas zero, como [1, 0, 0, 0, 0, 0, 3, 0, 0, 0]. Armazenar todos esses zeros em uma lista desperdiça memória, então os programadores costumam usar dicionários para manter o controle apenas das entradas diferentes de zero. Por exemplo, o vetor mostrado anteriormente seria representado como {0: 1, 6: 3}, porque o vetor que ele pretende representar tem o valor 1 no índice 0 e o valor 3 no índice 6.

Escreva uma função que converte um vetor esparso em um dicionário que o represente conforme explicado na descrição. A função deve ter a assinatura abaixo:

def converte_vetor(vetor: list) -> dict:

Um exemplo de execução da função está definido na tabela abaixo.

Entrada	Saída
converte_vetor([1, 0, 0, 0, 0, 0, 3, 0, 0, 0])	{0: 1, 6: 3}

3. A soma de dois vetores é apenas a soma de seus elementos. Por exemplo, a soma de [1, 2, 3] e [4, 5, 6] é [5, 7, 9]. Escreva uma função que pega dois vetores esparsos armazenados como dicionários e retorna um novo dicionário representando sua soma. A função deve ter a assinatura abaixo:

def adiciona_vetores(vetor1: dict, vetor2: dict) -> dict

Um exemplo de execução da função está definido na tabela abaixo.

Entrada	Saída
adiciona_vetores({0: 1, 6: 3}, {0: 2, 6: 4})	{0: 3, 6: 7}
adiciona_vetores({0: 1, 6: 3}, {3: 2, 8: 4})	{0:1, 3:2, 6:3, 8:4}