**Programa IT Academy – Processo Seletivo – Edição #18**

**Nome Completo: Bernardo Pacheco Fiorini**

**E-mail: bernardo.fiorini@edu.pucrs.br**

# **Etapa 1 – Questões de lógica**

Esta seleção possui 15 questões de lógica de caráter eliminatório. As questões são apresentadas no formulário de Exercício Técnico e devem ser respondidas no próprio formulário online, que deverá ser acessado através do link a seguir: https://forms.gle/yZtVcv1b5fCgScLBA

# **Etapa 2**

**RESUMO DA SOLUÇÃO**

Primeiramente escolhi a linguagem Python pois sabia que seria mais fácil de manipular arquivos “.csv”. Com o os arquivos em mãos, decidi criar mais 2 arquivos para organizar o código, “Itens.csv” e “Caminhoes.csv”, os quais contém as informações de peso do item, peso do caminhão, preço do caminhão em km/L. Para descobrir a distância entre duas cidades utilizei os index das cidades que o usuário digita, e devolvo a posição dela na matriz das distancias, e, para saber o custo total, multiplica a distância com o preço do caminhão que o usuário digitar.

Na segunda funcionalidade o usuário digita uma sequência de cidades, os itens que deseja transportar, e a quantidade de itens e deve retornar o custo, distância, itens etc. e armazena em diferentes listas. Tendo em mãos as listas dos respectivos valores de cada item, criado no início do programa, é possível obter qualquer informação buscando o index dela. (exemplificado no código a seguir no relatório). Então para imprimir o peso total, ele faz uma repetição das quantidadesdeitens[i] x preços[i] x distanciatotal. Ao longo do programa, realizo os cálculos utilizando alguns “for” para obter a distância entre cada parada, peso unitário, quantidade de caminhões necessários. E por fim, na opção 2, adiciono todas as listas que criei para a rota e adiciono na matriz “rotas”

Na terceira funcionalidade, o objetivo era imprimir todos os resultados obtidos, adicionando custo por parada, custo por km/L, número de veículos. Para isso, utilizei o método “.join” para ter um fácil acesso das listas das rotas que eu precisava, por exemplo: .join(rota[‘cidades’]) o programa recebe todas as cidades que o transporte iria passar.

**TESTES (aqui você deverá colar capturas de tela de todas as funcionalidades desenvolvidas e realizar comentários, use o espaço que julgar necessário)**

**FUNCIONALIDADE 1 – Consultar trechos x modalidade**

**Graphical user interface, text

Description automatically generated with medium confidence**

**Primeiramente, digitamos a opção 1, qualquer outro caractere que não está no menu irá resultar em uma mensagem de erro:**

****

**Ao prosseguir o programa ira abrir o input() para o usuário digitar o nome de duas cidades, podendo ser em letra maiúscula ou minúscula:**

****

**Se a qualquer das duas cidades não existir, o programa irá avisar e voltar para a pergunta:**

**Graphical user interface, text, application, website

Description automatically generated**

**Após isso, o usuário precisa determinar qual transporte irá utilizar:**

**Text

Description automatically generated**

**Por fim, imprime a distancia e o custo do trecho:**

****

**Para a implementação da funcionalidade 1 foi muito simples, utilizando o “import csv” em Python, pude criar uma matriz com todas as distancias. E buscando elas a partir de uma lista de todas as cidades.**

with open('DNIT-Distancias.csv', *newline*='') as csvfile: #Abre o .csv das cidades/distancias

dist\_reader = csv.reader(csvfile, *delimiter*=';') #Separo cada cidade pelo caracter ';'

cidades = next(dist\_reader) # Define que a primeira linha do arquivo sao as cidades

distancias = [] # Cria uma matriz das distancias

for linha in dist\_reader: #Leio as distancias e armazeno em "distancias"

distancias.append(linha)

**Com isso, testa se a cidade digitada existe na lista “cidades[]” e com o “.index(cidade1)” consegue-se buscar uma determinada distância[i][j]**

cidade1 = input(f"Digite o nome da primeira cidade: ").upper().strip() #Utilizo o "upper()" para digitar tudo em maiusculo, para facilitar a busca na lista

if cidade1 in cidades: #Aqui ele verifica se a cidade1 existe na lista do .csv

cidade2 = input(f"Digite o nome da segunda cidade: ").upper().strip() #Utilizo o "strip()" para descartar os espaços em branco

while True:

if cidade2 in cidades: #Aqui ele verifica se a cidade2 existe na lista do .csv

indice1 = cidades.index(cidade1) #Declaro os indices das 2 cidades na

indice2 = cidades.index(cidade2)

distancia = distancias[indice1][indice2] #E busco a distancia de acordo com os indices das cidades

**Seguindo para o código na parte das opções de transporte, utilizo uma condição para imprimir o desejado, para caminhão de pequeno porte, por exemplo:**

if menu1 == 1: #Caso a opcao seja Caminhão de pequeno porte

valor = float(preco[0])\*int(distancia) #Multiplica o preco(em float) com a distancia(devo transforma-la em int) com os seus index

print("De "+cidade1+ " para "+cidade2+", utilizando um "+caminhao[0]+", a distância é de "+distancia+"km e o custo será de R$"+str(valor))

print("") #Adiciono um espaço em branco

break

**Vale lembrar que para imprimir o valor é necessário transformá-lo em String.**

**FUNCIONALIDADE 2 – Cadastrar transporte**

**Graphical user interface

Description automatically generated with medium confidence**

**Ao escolher a opção 2, o usuário terá que digitar no mínimo duas cidades.**

**A picture containing graphical user interface

Description automatically generated**

**O programa poderá ler se estiver em maiúsculo, minúsculo, com espaços ou não. Caso a cidade não exista, ou digitar apenas uma cidade, o programa imprime:**

**Graphical user interface, text

Description automatically generatedGraphical user interface, text, application

Description automatically generated**

**Seguindo, o usuário precisa escolher os itens que deseja levar para a o seu transporte, digitando com virgulas o respectivo item:**

**Graphical user interface

Description automatically generated with medium confidence**

**No primeiro momento, precisa-se digitar a quantidade total dos itens que deseja transportar em toda a viagem:**

**Text

Description automatically generated**

**Após isso, digite a quantidade de itens que deseja descarregar em alguma cidade:**

**Text

Description automatically generated**

**No caso de digitar um valor maior que o valor total de qualquer item, o programa irá avisar:**

**Graphical user interface, text, website

Description automatically generated**

**Finalmente, o programa devolve um relatório, mostrando a quantidade de caminhões que serão necessários para o transporte, a distância total, a distancia de cada parada, preço total e o custo unitário médio:**

**Text

Description automatically generated**

**Para facilitar a implementação, criei mais 2 arquivos “.csv”, “caminhões.csv”e  
“itens.csv” contendo todas as informações de nome do caminhão, peso do caminhão, nome do item, e peso do item:**

with open('Caminhoes.csv', *newline*='') as csvfile: #Abre o .csv dos caminhoes

dist\_reader = csv.reader(csvfile, *delimiter*=',') #Dessa vez, separo cada informacao pelo caracter ';'

caminhao = next(dist\_reader) #Le a proxima linha e armazena na lista "caminhao"

pesoC = next(dist\_reader) #Le a proxima linha e armazena os pesos de cada na lista "pesoC"

preco = next(dist\_reader) #Le a proxima linha e armazena os preços na lista "preco"

with open('Itens.csv', *newline*='') as csvfile: #Abre o .csv dos itens

dist\_reader = csv.reader(csvfile, *delimiter*=',') #Separa cada informacao pelo caracter ','

item = next(dist\_reader) #Le a proxima linha e armazena os nomes dos itens em "item"

peso = next(dist\_reader) #Le a proxima linha e armazena os pesos dos respectivos itens na lista "peso"

**Ao digitar as cidades seguidos por vírgula:**

transporte = input()

listatransporte = [transporte.strip().upper() for transporte in transporte.split(",")] #Aqui ele recebe as cidades separadas por vírgula, usando o strip() para descartar os espaços em braco e o split(","")

**A listatransporte[] organiza e padroniza as cidades para o código seguinte:**

**Executando um teste para se caso a cidade não existir:**

for cidade in listatransporte:

if cidade not in cidades: #Se qualquer uma cidade nao existir, ele nao avança

print("")

print("Uma ou mais das cidades não existe(m)")

print("")

break

elif len(listatransporte)<2: #Se digitar apenas uma cidade, ele nao avança

print("")

print("Digite no mínimo duas cidades")

print("")

break

**Ao digitar os itens que deseja transportar o código seguinte organiza uma lista dos itens que serão necessários.**

itens = input()

listaitens = [int(itens.strip()) for itens in itens.split(",")] #Organiza a lista dos index dos itens

**Agora eu preciso criar uma lista que irá receber o valor dos itens, ou seja, se eu escolhi 5 geladeiras e 5 celulares, irá criar uma lista [5,5]. Nota-se que utilizo um for percorrendo a listaitens[] para imprimir.**

**Aqui eu decremento 1 no listaitens pois no menu é “1-Celular” mas na verdade é necessário o index 0, então, item[0]**

**\*** item[listaitens[i]-1].

**ex: Geladeira: (valor que o usuário digitar) Celular: (valor que o usuário digitar)**

**Também crio uma lista temporária para manipular a quantidade total de itens que serão entregues em cada parada.**

qtdTotal = [] #Cria uma lista que irá receber as quantidades dos respectivos itens

for i in range(len(listaitens)):

qtdTotal.append(int(input(item[listaitens[i]-1]+ ":")))

qtdtemp = [] #Lista temporária que irá diminuir as quantidades dos itens que ja foram entregues

for i in range(len(qtdTotal)):

qtdtemp.append(qtdTotal[i])

**Logo após, preciso criar uma matriz que carregará a quantidade de itens de cada parada. Então no “qtdparadas” será [[itens da parada1][itens da parada2]]**

qtdparadas = [] #Cria uma matriz dos valores de cada parada

**Passando por todas as paradas, o usuário digita um valor de celulares, que será decrementado pelo valor total do item. Se o valor digitado for maior que a quantidade total de celulares, aparece a mensagem que ultrapassou o limite.**

for i in range(len(listatransporte)-1):

print("De " + listatransporte[i] + " até " + listatransporte[i+1]+ " Quantos itens você quer levar?")

qtd = [] #Cria uma lista vazia para receber os valores de cada parada

for i in range(len(listaitens)): #Percorre a lista de itens

while True:

valor = int(input(item[listaitens[i]-1]+ ":"))

if (valor <= qtdtemp[i]): #Se o valor for menor ou igual ao total

qtd.append(valor) #Adiciona o valor na lista da parada

qtdtemp[i] = qtdtemp[i]-valor #Entao, sempre quando ele receber um valor, sera decrementado no item no qtdtotal

break

else: #Se o valor digitado for maior que a quantidade total, da erro

print("Limite ultrapassado! Digite outro valor")

print("")

qtdparadas.append(qtd) #Adicionou a lista de itens na parada no qtdparadas

**Assim, o transporte será cadastrado.**

**Para a distancia total do transporte utilizei um for para criar uma lista de todas as distancias, posteriormente, utilizo o sum() para a distância total**

for i in range(len(listatransporte)-1):

distanciat= int(distancias[cidades.index(listatransporte[i])][cidades.index(listatransporte[i+1])]) #Calcula a distancia de cada parada

distanciatotal.append(distanciat) #Adiciona na lista a distancia de todas as paradas

distanciasoma = sum(distanciatotal)

**Utilizando a mesma lógica, calculei o peso total dos itens, criando uma lista e somando todos os elementos dela. Entretanto, aqui foi necessário manipular a matriz, por a “qtdparadas” está organizada como [parada][qtdItem]. Com a quantidade selecionada, multiplica com o peso do respectivo item.**

for j in range(len(qtdparadas[i])): #Calcula o peso total de todos os itens

print(item[listaitens[j]-1]+" x"+str(qtdparadas[i][j]))

pesot = float(peso[listaitens[j]-1])\*qtdparadas[i][j]

pesototal.append(pesot)

pesosoma = sum(pesototal)

**Para calcular a quantidade necessária de caminhões, utilizei o peso total e diminui com o valor dos pesos dos caminhões.**

**A lista dos pesos dos caminhões é [10000,4000,1000] então sempre que o peso total for ou igual a um dos pesos da lista, ele incrementa em um array.**

caminhoesn = [0,0,0] #Cria uma lista

for i in range(len(pesoC)): #For com a quantidade de tipos de pesos

while pesosoma >= float(pesoC[i]): #Se for maior ou igual ao peso do index ele incrementa em caminhoesn

caminhoesn[i] = caminhoesn[i]+1

pesosoma = pesosoma -float(pesoC[i]) #Diminiu o valor do peso total

if pesosoma < float(pesoC[2]) and pesosoma!=0: #Aqui ele pega o resto, se tiver um valor menor que 1000, ele incrementa 1

caminhoesn[i] = caminhoesn[i]+1

pesosoma = pesosoma -float(pesoC[i])

**Exemplo: peso = 15000**

**então peso - 10000 == [1,0,0]**

**peso - 4000 == [1,1,0]**

**peso - 1000 == [1,1,1]**

**Deverá ter: 1 Caminhão de grande porte**

**1 Caminhão de médio porte**

**1 Caminhão de pequeno porte**

**Para calcular o valor total de toda a viagem, preço unitário médio criei essa função, que cria uma lista dos valores e depois eu somo todos os elementos presentes nela:**

valorT=[] #Para calcular o valor total de toda a viagem

for i in range (len(caminhoesn)): #Percorre a lista dos caminhoes necessários

valor = caminhoesn[i]\*distanciasoma\*float(preco[i]) #E multiplica pelo preço do caminhao e a distancia total

valorT.append(valor) #Adiciona o valor na lista valotT

unit = float(sum(valorT)/sum(qtdTotal)) #Calcula o preco unitario medio

**FUNCIONALIDADE 3 – Dados estatísticos**

**Graphical user interface

Description automatically generated**

**Para acessar os dados estatísticos de cada rota, o usuário digita ‘3’. Caso for acessado sem nenhuma rota cadastrada, irá aparecer um erro:Graphical user interface, text, application, website

Description automatically generated**

**Caso alguma rota já tenha sido cadastrada, ele mostra o relatório:**

**Graphical user interface, text

Description automatically generated**

**Para facilitar a implementação da funcionalidade 3, no final do cadastro do transporte no menu2, adicionei todas as informações úteis em uma matriz de “rota”**

rota = {'cidades': listatransporte, 'itens': listaitens, 'qtdTotal': qtdTotal, 'valorT': sum(valorT), 'unit': unit, 'custokm': custokm}

rotas.append(rota)

**Na opção 3, verifica se existe uma ou mais rotas cadastradas:**

if len(rotas)>=1:

#codigo3

else:

print("Nenhum transporte foi cadastrado")

**#codigo3:**

**Percorrendo toda a lista de rotas, com o método “.join” consigo acessar todas as listas que declarei na opção 2. Para imprimir eu copiei a mesma logica da opção 2.**

for i, rota in enumerate(rotas):

print(f"Rota {i+1}: {' --> '.join(rota['cidades'])}") #Concatena as cidades registradas na rota

print(f"Itens transportados: {', '.join([item[k-1] for k in rota['itens']])}")

#Imprime os itens transportados

print(f"Quantidades transportadas, respectivamente: {', '.join([str(qtd) for qtd in rota['qtdTotal']])}") #Percorre a lista das quantidade de itens

print(f"Valor total R${', '.join([str(rota['valorT'])])}")print(f"Valor unitário R${', '.join([str(rota['unit'])])}")

print(f"Custo por Km R${', '.join([str(rota['custokm'])])}") #Imprime o custo por km

**AUTOAVALIAÇÃO**

Você concluiu a implementação de 100% das funcionalidades solicitadas?

(X) Sim ( ) Não

Para as 3 principais funcionalidades solicitadas, como você avalia a sua solução?

Marque um ‘X’.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Inexistente/ Insuficiente | Pouco satisfeito(a) | Satisfeito(a) | Muito satisfeito(a) |
| Funcionalidade 1 |  |  |  | X |
| Funcionalidade 2 |  |  | X |  |
| Funcionalidade 3 |  |  |  | X |

**Principais dificuldades**

A principal dificuldade foi a implementação da funcionalidade 2, pois nela, foi necessário armazenar muitos valores em muitas listas diferentes. Por exemplo, no cálculo das quantidades de caminhões necessários para o transporte, tive problema para diminuir os pesos dos caminhões com o “pesoTotal”, pois sempre sobrava um valor que não contabilizava na lista de caminhões necessários. Para isso, tive que pegar um caderno e escrever um pseudocódigo para me orientar e descobrir o erro.

**Desempenho Geral**

Na primeira funcionalidade obtive um desempenho muito bom, conseguindo executar em poucos minutos, criando uma matriz das distâncias e buscar a partir do index das cidades resultou em uma aplicação muito funcional.

Na segunda funcionalidade tive um pouco mais de trabalho, pois, primeiramente, organizei as minhas ideias em um caderno para não me perder no raciocínio. Por fim, obtive um resultado satisfatório, porém, poderia ter feito de maneira mais eficiente e funcional. Utilizando diferentes tipos de itens e quantidades de paradas, descobri que o custo total poderia ser diferente em alguns casos. Imagino que eu precisaria criar uma matriz com todas as possibilidades possíveis de custos de caminhões para levar a carga comparando com a distância de cada parada, porém, não consegui implementar.

Na ultima funcionalidade não tive muitas dificuldades, pois eu já havia trabalhado com esse tipo de informação em um trabalho da faculdade, então, utilizando o “.join”, pude imprimir as informações desejadas.

Obrigado por participar deste processo seletivo.  
Salve o documento em PDF com o seu nome completo.