Nome: Kaique Teixeira dos Anjos

Funções para correção dos dados:

Função ler_banco:

```
def ler_banco(arquivo):
    try:
        with open(arquivo, 'r', encoding='utf-8') as banco_json:
            banco = json.load(banco_json)
    except FileNotFoundError:
        print(f'O arquivo "{arquivo}" não foi encontrado nesse
diretório')
    return banco
```

Essa função foi criada para receber um arquivo json, passando seu caminho como um parâmetro da função, assim podendo usar essa função tanto para abrir o broken-database.json quanto o saída.json e os retornando como uma lista de dicionários. Usei a biblioteca json para utilizar a função load que carrega o arquivo json para um dado em python.

Função corrigir_nomes:

```
def corrigir_nomes(banco):
    entrada = 'øß¢æ'
    saida = 'obca'
    tabela = str.maketrans(entrada, saida)

    for dado in banco:
        decod = dado['name']
        dado['name'] = decod.translate(tabela)
    return banco
```

Nessa função, primeiro observei o banco e percebi um padrão nos dados corrompidos, dessa forma criei uma tabela com dados de entrada e saída (ambos com dados que correspondiam um ao outro), usei o maketrans para receber os dados para tradução e depois o translate para executar a tradução dentro de um for que analisava cada dicionário de produto e no campo ‘name’, se fosse encontrado um caractere igual a um caractere de entrada, era substituído pelo seu correspondente em saída, e no fim retorna o banco com os nomes corrigidos.

Função corrigir_precos:

```
def corrigir_precos(banco):
    for dado in banco:
        preco = str(dado['price'])
        preco = float(preco)
        dado['price'] = preco
    return banco
```

Com um for caminhando por toda a lista e analisando o campo ‘price’ do dicionário, primeiro se convertia o valor de ‘price’ para uma string e após isso era convertido para um float e gravado novamente na mesma posição que se encontrava. Obs.: Converti todos os preços para string primeiro, mesmo alguns já estarem em float, para não haver necessidade de usar um if para verificar se era string ou não, assim consegui deixar o código mais enxuto.

Função corrigir_quantidade:

```
def corrigir_quantidade(banco):
    nBanco = list()
    nDado = dict()
    for dado in banco:
        if 'quantity' not in dado:
            dado['id'] = dado['id']
            dado['name'] = dado['name']
            dado['quantity'] = 0
            dado['price'] = dado['price']
            dado['category'] = dado['category']

    for dados in banco:
        nDado['id'] = dados['id']
        nDado['name'] = dados['name']
        nDado['quantity'] = dados['quantity']
        nDado['price'] = dados['price']
        nDado['category'] = dados['category']
        nBanco.append(nDado.copy())
    return nBanco
```

Nessa função criei uma lista nova para receber a lista que estava todos os dados, mas com a diferença que se não fosse encontrado a key ‘quantity’ em algum desses dados, ela seria incluída com o valor 0, assim como foi falado no problema e com ela adicionada, era gravado na nova lista todo o dicionário, assim mantendo o quantity na mesma ordem que estavam os outros.

Função exportar_banco:

```
def exportar_banco(banco):
    with open('saida.json', 'w', encoding='utf-8') as saida_banco:
        json.dump(banco, saida_banco, indent=4, ensure_ascii=False)
```

Semelhante ao ler_banco(), essa função utiliza de outra função da biblioteca json, que é o dump que gera um arquivo json com os dados passados nele que nesse caso é o banco que originalmente estava corrompido, mas que agora foi corrigido após passar por todas as funções acima, no dump coloquei o indent=4 para ele deixar identado a lista e o ensure_ascii=False, para que quando fosse exportado, as letras com acentuação não ficassem em código ascii, mas sim no padrão utf-8.

Funções para validação do banco corrigido:

Função listagem_produtos:

```
def listagem_produtos(banco):
    listaOrdenada = sorted(banco, key=lambda k: (k['category'],
k['id']))
    print('-' * 130)
    print('LISTA ORDENADA POR CATEGORIA E ID:'.center(130))
    print('-' * 130)
    for prod in listaOrdenada:
        print(f'NOME: {prod["name"]:.<80} ID: {prod["id"]}  CATEGORIA:
{prod["category"]}')
```

Como no problema pede-se que seja ordenado por dois fatores, usei o sorted(), passando dentro dele o parâmetro key como lambda com dois Keys diferentes (‘category’ e ‘id’), assim ele primeiro vai deixar em ordem alfabética pela categoria e nos casos que a categoria for igual, ele ordena pelo número do id, printando tudo usando um for que andava por todos os dados da lista e imprimia apenas o nome do produto, id e categoria. Obs.: apesar de não ser solicitado, printei o id e categoria para ficar mais fácil de visualizar a ordenação.

Função estoque_categoria:

```
def estoque_categoria(banco):
    dictCategoria = {'Panelas': 0, 'Eletrodomésticos': 0,
'Eletrônicos': 0, 'Acessórios': 0}
    for k in dictCategoria.keys():
        for dado in banco:
            if dado['category'] == k:
                dictCategoria[k] += dado['quantity']

    print('-' * 35)
    print('VALOR DO ESTOQUE POR CATEGORIA:'.center(35))
    print('-' * 35)
```

```
    for k, v in dictCategoria.items():
        print(f'{k:.<20}{v:>3}')
```

Nessa função para deixar mais enxuto, optei por criar um dicionário para ir armazenando o total por categoria toda vez que o valor na key ‘category’ fosse igual a key desse dicionário que criei para armazenar os totais, assim não precisando fazer um if para cada categoria existente, e tornando o print desses totais mais simples.

Programa principal:

```
try:
    banco = ler_banco('broken-database.json')
except UnboundLocalError:
    print('Não foi possível ler o arquivo!')
else:
    banco = corrigir_nomes(banco)
    banco = corrigir_precos(banco)
    banco = corrigir_quantidade(banco)
    exportar_banco(banco)
    banco = ler_banco('saida.json')
    listagem_produtos(banco)
    estoque_categoria(banco)
```

Aqui apenas chamei todas as funções para serem executadas.

O porquê de usar Python:

Usei o python por três razões principais, que foi pelo fato dele ser bem simples, então consigo produzir mais em menos tempo, para trabalhar com dados ele é muito potente, pois conversões, fatiamentos, busca de dados e ordenação são feitos de maneira bem simples, fora as bibliotecas que podem ser importadas como foi o caso do json, e como estou aprendendo Python recentemente, preferi usa-lo para poder praticar o que aprendi e se sou capaz de resolver problemas assim usando a linguagem

Tratamentos feitos para evitar bugs e outros:

Usei alguns try e except para prevenir mensagens de erro de exceção como o caso de não encontrar o arquivo no diretório que será executado o código, caso ocorra, é mostrado que não foi encontrado o arquivo e que não é possível concluir as funções seguintes, encerrando o programa por ali mesmo. Fora as condições e uso de fors para evitar loops e falhas na hora de tratar os dados, como foi o caso de converter todos os preços para string para depois passar para float, assim evitando código a mais e possíveis brechas para não se converter os preços.