Descrição das funções encontradas em ‘resolucao.js’:

read_file_broken():

Uso do módulo fs para leitura síncrona (método fs.readFileSync, com codificação UTF-8) do arquivo ‘broken-database.json’, com declaração try/catch para tratativa de possíveis erros. Também nesta função, é feita a verificação se há conteúdo no arquivo; caso o mesmo esteja vazio, é retornada a mensagem “Seu arquivo está vazio.” e, havendo conteúdo no arquivo é realizada a conversão deste conteúdo para o tipo objeto (método JSON.parse()), sendo possível realizar toda manipulação das demais funções, através do array ‘data_obj'.

fix_name():

Loop em ‘for’; tem como limite o número total de objetos do arquivo (uso da propriedade ‘length’) para percorrer a String ‘name’ de cada objeto do arquivo e uso do método replace() para realizar a substituição dos caracteres citados na lista, através do uso de RegEx, e sobrescrever cada valor ‘name’ de cada objeto.

fix_price():

Loop em ‘for’ utilizando as mesmas expressões do loop encontrado na função ‘fix_name’. Ao percorrer o loop é feita a conversão do tipo String para o tipo number; neste caso, foi utilizada a função parseFloat() pois o ponto flutuante é mais ideal para representar um valor moeda.

fix_quantity():

Uso do loop for para percorrer todos os objetos do arquivo. Dentro do loop foi realizado um if para verificar se a chave ‘quantity’ é existente. Quando o retorno dessa condição for ‘True’ (ou seja, a chave em questão não é localizada no objeto) é feito um novo preenchimento deste objeto, acrescentando a chave ‘quantity’ com valor ‘0’.

export_corrected_file():

Uso do método JSON.stringify para converter os valores obtidos da variável ‘data_obj’ para String JSON e uso do módulo fs para escrita síncrona (método fs.writeFileSync, com codificação UTF-8) do arquivo ‘saida.json’, realizando a exportação dos dados tratados (após a chamada das funções de correção) para o arquivo nomeado (saida.json). Também é feita a declaração try/catch para tratativa de possíveis erros.

names_validation():

Uso do método sort(), passando como parâmetro uma função de comparação para possibilitar a ordenação dos ‘ids’ (‘return’ definido através do uso de dois operadores ternários, retornando os possíveis valores de retorno do método sort, -1, 1 ou 0); após definir este primeiro critério de ordenação, da mesma forma foi realizada a ordem das categorias, em ordem alfabética. Para a impressão dos valores ‘name’, foi realizado um for para percorrer todos os objetos e printar apenas o nome de cada valor ‘name’.

Values_validation():

Para deixar a impressão de valores interativa a inclusão de novas categorias, optei por ordenar os objetos por valor ‘category’ para então iniciar a busca por quantidades e valores de cada categoria. A busca se inicial em um loop for, responsável por controlar o índice do objeto com filtro para o ítem ‘category’. Posteriormente, a variável ‘categoria_atual’, inicializada com o valor vazio, é comparada a categoria do objeto índice ‘0’ (ditado pelo ‘for’), e, quando são

diferentes, a ‘categoria_atual’ passa a assumir o valor do ítem ‘category‘ do objeto em questão. Então, é iniciado um novo loop, responsável por controlar o índice do objeto com filtro para os ítens ‘quantity’ e ‘price’, seguido de um novo ‘if’, onde é checado se o valor de ambas as variáveis são iguais; sendo o retorno ‘True’, é feita a operação de multiplicação entre a quantidade e preço do produto referente àquele índice, e, em cada passo em que ocorre a validação deste segundo ‘if’, o valor é incrementado a variável ‘valor_final’. Ao finalizar a condição do segundo ‘for’ citado, o valor é informado na tela e o valor ‘0’ é atribuído a variável ‘valor_final’. Então, é iniciado uma busca pela próxima ‘category’ (primeiro for citado), seguindo os passos citados acima. O processo é finalizado apenas após o primeiro for percorrer todos os objetos do arquivo.