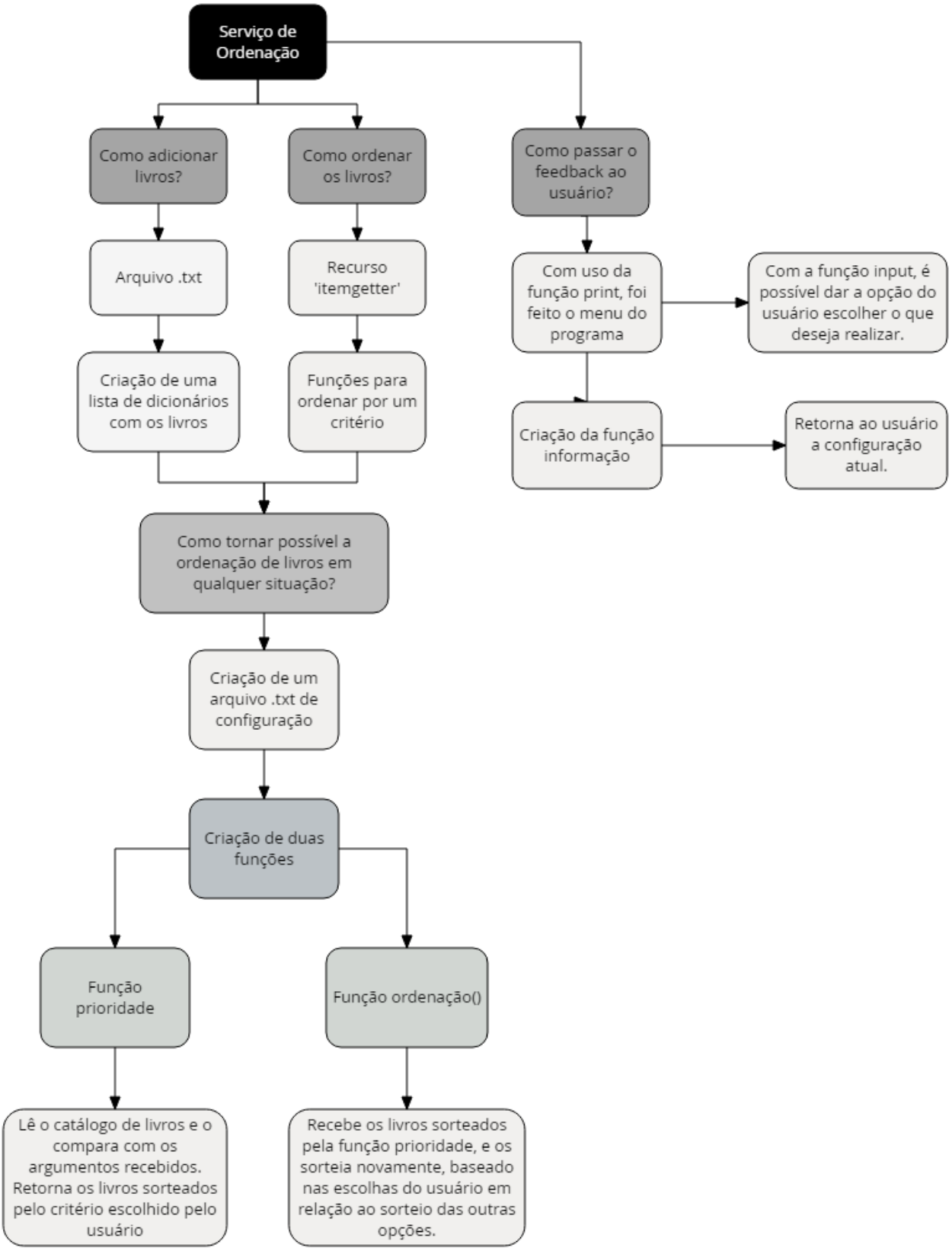
SumOne Programming Challenge: Serviço de Ordenação
Autora: Bruna Carpes de Freitas
LInguagem: Python 3.7 64 bits

Neste documento, explico a arquitetura e escolhas do meu projeto. Abaixo, segue um fluxograma para tornar a explicação mais fácil e visual.



As primeiras perguntas feitas ao iniciar a arquitetura foram: “Como adicionar livros?” e “Como ordenar os livros?”.

Para resolver a questão “Como adicionar livros?”, um arquivo .txt externo provou-se ser a melhor forma de adicionar ou remover livros, afinal, ele tornaria a organização mais prática, trazendo ao usuário a possibilidade de adicionar livros sem precisar executar o programa. Foi criado então o arquivo ‘catalogo.txt’. Neste arquivo, foram inseridos os livros teste disponibilizados na página do desafio. Agora, precisava-se criar uma função para ler os livros adicionados neste arquivo e armazená-los em uma variável dentro do programa. Assim foi criada a função livros(). Ela lê cada linha do arquivo e, para cada livro, ela cria um dicionário com as seguintes *key*: índice, título, autor e ano. Após ler e criar os dicionários, a função armazena todos os dicionários em uma lista chamada catalogo.

Agora que a primeira questão fora resolvida, era necessário resolver a questão seguinte: “Como ordenar estes livros?”. Para começar, decidiu-se criar funções que fizessem a ordenação básica, pré-determinada dos livros. Para ordenar os livros, não importa por qual critério, título, autor ou ano, foi utilizada uma ferramenta chamada *itemgetter*. Essa ferramenta, disponibilizada pela biblioteca *operator*, busca em um dicionário os valores correspondentes a uma determinada *key*. Quando combinada com a ferramenta *sorted*, ela torna possível a ordenação de uma lista de dicionários por qualquer *key* desejada. Nesta situação, caso deseje-se sortear a lista pela *key* ‘títulos’, basta atribuir à ferramenta *itemgetter* a *key* ‘títulos’ e será possível sortear a lista.

Agora era necessário tornar o programa mais abrangente porém *clean*. Funções engessadas não agradam a ninguém, então mostrou-se necessário criar algo que tornasse o programa mais personalizável.

Foi criado um arquivo para permitir a configuração a externa das prioridades e ordens de ordenação. A leitura dele funciona da mesma forma que a leitura do arquivo catalogo.txt, com uma função chamada config, porém, ao invés de armazenar os valores em uma lista de dicionários, armazena estes em uma lista simples. Neste arquivo de configuração, o usuário deve inserir a preferência de ordenação (título, autor ou ano) e a ordem da ordenação para o título, autor e ano, respectivamente. A ordem de ordenaçao corresponde a um número de 0 a 2, sendo: 0 - indiferente, 1 - ascendente e 2 - descendente. A ordenação indiferente deve ser atribuída somente às ordenações secundária e terciária. A ordenação do critério preferencial deve ser escolhida obrigatoriamente.

Tendo a configuração escolhida pelo usuário, era necessário transformar as três funções de ordenação, que eram engessadas, em uma função que funcionasse para as três possíveis preferências de ordenação. Foi criada então a função prioridade(cri, orT, orAu, orAn). Esta função recebe 4 argumentos: os valores da lista criada pela função config: cri sendo a preferência de ordenação; orT, orAu e orAn sendo o número para ordenação do título, autor e ano, respectivamente. Comparando os argumentos, a função encontra qual foi a escolha do usuário para ordenação primária da lista. Por exemplo, caso o usuário tenha escolhido título como preferência de ordenação e tenha determinado que queria que a ordenação por título fosse descendente, a função prioridade encontraria esta combinação, sortearia a lista catalogo pelo título de forma descendente e retornaria este valor para uma nova lista chamada PriC.

Para as ordenações secundária e terciária, foi criada uma segunda função chamada ordenação. Ela pega a lista PriC, que foi retornada pela função prioridade e a sorteia novamente, utilizando as escolhas secundária e terciária do usuário. É uma função que se adapta as escolhas, graças as variáveis condicionais que possuem atribuição dinâmica de acordo com a entrada do usuário. Reutilizando o exemplo anterior, caso escolha título como preferência de ordenação, as ordenações secundária e terciária serão ano e autor, respectivamente, caso o usuário não tenha optado por “indiferente”.

Foi criada uma interface simples para comunicação com o usuário. Através da função *print*, informa-se as opções que este possui dentro do programa. Através de uma chamada da função *input*, o usuário pode escolher o que deseja fazer. Os resultados de ordenação são

passados através da utilização da função *tabulate*, uma biblioteca para Python que transforma em tabela dados de listas, tuples ou dicionários. Também foi criada uma função simples, chamada informação. Esta função passa ao usuário quais as configurações atuais do arquivo config. Caso o usuário deseje finalizar o programa, basta digitar ‘fim’.

Para facilitar a vida do usuário, foi criado um arquivo executável. Alterações no código não alteram o executável, mas alterações nos arquivos de texto, sim. O arquivo executável também garante a independência da plataforma Python ou de qualquer biblioteca utilizada.