INSTITUTO FEDERAL
Catarinense
Campus Blumenau

Algoritmos

Bacharelado em Ciência da Computação
Prof. Paulo César Rodacki Gomes

Lista de exercícios — Alg — 06
Exercícios sobre Strings, Tuplas e Listas em Python

Estes exercícios devem ser entregues no Google Classroom. Para cada um dos exercícios, crie um arquivo fonte Python com o respectivo nome de acordo com a seguinte regra: `SUASINICIAIS-Alg-06-Ex-num.py`. Por exemplo, se o professor resolvesse o exercício número 3, o nome do arquivo seria `PCRG-Alg-06-Ex-03.py`.

Introdução

As listas ajudam os programadores a gerenciar grandes quantidades de dados, permitindo que vários (ou mesmo muitos) valores sejam armazenados em uma variável. Isso torna prático resolver problemas maiores que envolvem muitos valores de dados. Para resolver os exercícios neste capítulo, você deve esperar ter que realizar as seguintes tarefas:

- Criar uma variável que contenha uma lista de valores
- Modificar uma lista anexando, inserindo, atualizando e excluindo elementos
- Pesquisar um valor em uma lista
- Exibir alguns ou todos os valores em uma lista
- Escrever uma função que tenha uma lista como parâmetro
- Escrever uma função que retorna uma lista como seu resultado

Questões:

1. **Ordem Crescente**. Escreva um programa Python que leia números inteiros (do usuário) e os armazena em uma lista. Seu programa deve continuar lendo inteiros até que o usuário entre com o valor 0 (zero). Então, o programa deve exibir em ordem **crescente** todos os números digitados pelo usuário sem incluir o valor 0, com um valor em cada linha impressa. Obs.: você pode implementar o algoritmo de ordenação BubbleSort ou usar o método **sort** ou a função **sorted** para ordenar a lista.

2. **Ordem decrescente.** Escreva um programa Python que leia números inteiros (do usuário) e os armazena em uma lista. Seu programa deve continuar lendo inteiros até que o usuário entre com o valor 0 (zero). Então, o programa deve exibir em ordem **decrescente** todos os números digitados pelo usuário sem incluir o valor 0, com um valor em cada linha impressa.

3. **Removendo extremos.** Ao analisar os dados coletados como parte de um experimento científico, pode ser desejável remover os valores mais extremos antes de realizar outros cálculos. Escreva uma função que tenha uma lista de valores e um número inteiro não negativo, n, como seus parâmetros. A função deve criar uma nova cópia da lista com os n maiores elementos e os n menores elementos removidos. Em seguida, ele deve retornar a nova cópia da lista como o único resultado da função. A ordem dos elementos na lista retornada não precisa coincidir com a ordem dos elementos na lista original.

   Escreva um programa principal que demonstre sua função. Sua função main deve ler uma lista de números do usuário e remover os dois maiores e os dois menores valores dela. Exiba a lista com os extremos removidos, seguido pela lista original. Seu programa deve gerar uma mensagem de erro apropriada se o usuário inserir menos de 4 valores.

4. **Evitando duplicatas**. Crie um programa Python que leia palavras do teclado até que o usuário forneça uma palavra vazia (somente a tecla enter). Depois disso, seu programa deve

mostrar somente uma vez cada palavra digitada pelo usuário. Ou seja, se o usuário forneceu mais de uma vez a mesma palavra, ela só poderá ser exibida uma vez. As palavras devem ser exibidas na mesma ordem em que foram digitadas. Por exemplo, se o usuário digitar:

```
Primeira
Segunda
Primeira
Terceira
Segunda
```

Então seu programa deve exibir:

```
Primeira
Segunda
Terceira
```

5. **Negativos, zeros e positivos.** Escreva um programa Python que leia números inteiros até que uma linha em branco seja fornecida pelo usuário (se ele digitar somente enter). Depois que todos os inteiros tiverem sido lidos, seu programa deve mostrar todos os números negativos, seguidos por todos os zeros e depois todos os números positivos. Dentro de cada grupo os números devem ser exibidos na ordem em que foram fornecidos pelo usuário. Por exemplo, se o usuário forneceu os valores 3, -4, 1, 0, -1, 0 e -2, então seu programa deve exibir os valores -4, -1, -2, 0, 0, 3 e 1, cada um em uma linha.

6. **Lista de divisores.** Um divisor de um numero inteiro n é um número inteiro positivo menor que n, tal que divida n em partes inteiras (divisão exata, sem resto). Escreva uma função Python que calcula todos os divisores de um número inteiro positivo. A função deve retornar uma lista contendo todos os divisores. Escreva uma função main para demonstrar o funcionamento da sua solução, a função main deve ler um número do usuário e imprimir todos os seus divisores.

7. **Números perfeitos.** Um número inteiro positivo n é chamado de número perfeito se a soma de todos os divisores de n é igual a n. Por exemplo, 28 é um número perfeito porque seus divisores são 1, 2, 4, 7 e 14; e 1+2+4+7+14=28. Escreva uma função para verificar se um número é perfeito. A função deve receber somente um número inteiro positivo e retornar True se ele for perfeito ou False caso contrário. Escreva uma função main para identificar e imprimir todos os números perfeitos de 1 a 10.000. Obs.: você pode usar o código do exercício anterior para lhe ajudar nesta tarefa.

8. **Somente palavras**. Neste exercício você deve criar uma função em Python que recebe um texto em uma string e retorna uma lista somente com as palavras, sem espaços ou símbolos de pontuação. Por exemplo, se a string for: “`Beleza! Este é um ótimo exemplo, você não acha?`”, sua função deveria retornar a seguinte lista: [“`Beleza", “Este", “é", “um", “ótimo", "exemplo", “você", “não", “acha”]`. Escreva uma função main que demonstre o funcionamento da sua solução.

9. **Abaixo e acima da média.** Escreva um programa que leia números do usuário até que uma linha em branco seja inserida. Seu programa deve exibir a média de todos os valores inseridos pelo usuário. Em seguida, o programa deve exibir todos os valores abaixo da média, seguidos por todos os valores médios (se houver), seguidos por todos os valores acima da média. Uma mensagem apropriada deve ser exibida antes de cada lista de valores.

10. **Formatando uma lista**. Quando escrevemos uma lista em português, geralmente separamos os itens por vírgula e colocamos a conjunção “e" entre os dois últimos itens, a não ser que a lista só tenha um item. Considere as listas abaixo:

```
maçãs
maçãs e laranjas
maçãs, laranjas e bananas
maçãs, laranjas, bananas e limões
```

Escreva uma função que receba como parâmetro uma lista de strings e retorne uma única string contendo todos os itens da lista formatados conforme mostrado acima. Apesar dos exemplos acima terem no máximo 4 itens, sua função deve se comportar corretamente para listas de qualquer tamanho. Escreva uma função main demonstrando o funcionamento de sua função.

11. **Mega-sena.** Para ganhar o prêmio da mega-sena, é necessário acertar todos os 6 números em seu bilhete com os 6 números entre 1 e 60 sorteados pelo organizador da loteria. Escreva um programa que gere uma seleção aleatória de 6 números para um bilhete de mega-sena. Certifique-se de que não haja números repetidos entre os sorteados. Exiba os números em ordem crescente.

12. **A lista está ordenada?** Escreva uma função que determina se uma lista de valores está ou não em ordem de classificação (crescente ou decrescente). A função deve receber a lista como parâmetro e retornar True se a lista já estiver classificada. Caso contrário, ele deve retornar False. Escreva um programa principal que leia uma lista de números do usuário e use sua função para relatar se a lista é classificada.

    Observação: certifique-se de considerar estas questões ao concluir este exercício: Uma lista vazia está em ordem? E uma lista contendo somente um elemento?

13. **Contagem de elementos**. A biblioteca padrão de funções do Python possui um método chamado **count**, que determina quantas vezes um determinado valor ocorre em uma lista. Neste exercício você deve criar uma nova função chamada **countRange** que deve determinar e retornar a quantidade de elementos em uma lista que são maiores ou iguais a um valor mínimo e menores que um valor máximo. Sua função deve receber três parâmetros: a lista (de números), o valor mínimo e o valor máximo. Ela deve retornar um valor inteiro maior ou igual a zero. Escreva uma função main demonstrando o funcionamento de sua função.

14. **Precedência de operadores.** Escreva uma função chamada **precedencia** que retorna um inteiro representando a precedência de um operador matemático. Uma string contendo o operador será passada para a função como seu único parâmetro. Sua função deve retornar 1 para + e -, 2 para * e / e 3 para ^. Se a string passada para a função não for um desses operadores, a função deve retornar -1. Inclua um programa principal que lê um operador do usuário e exibe a precedência do operador ou uma mensagem de erro indicando que a entrada não era um operador.

    Observação: Neste exercício, junto com outros que aparecem posteriormente nesta lista, usaremos ^ para representar a exponenciação. Usar ^ ao invés do operador ** do Python tornará esses exercícios mais fáceis porque um operador sempre será um único caractere.

15. **"Tokenização" de strings.** A tokenização é o processo de conversão de uma string em uma lista de substrings, conhecidas como *tokens*. Em muitas circunstâncias, uma lista de tokens é muito mais fácil de trabalhar do que a string original porque a string original pode ter espaçamento irregular. Em alguns casos, um trabalho substancial também é necessário para determinar onde termina um token e onde começo o próximo.

    Em uma expressão matemática, os tokens são itens como operadores, números e parênteses. Alguns tokens, como *****, */*, **^**, **(** e **)** são fáceis de identificar porque o token é um único caractere e nunca faz parte de outro token. Os símbolos **+** e **-** são um pouco mais desafiadores de tratar porque podem representar o operador de adição ou subtração ou podem ser parte de um token de número.

    Dica: Um sinal de + ou de - é um operador se o caractere (diferente de espaço em branco) imediatamente anterior fizer parte de um número ou se o caractere (diferente de espaço em branco) imediatamente antes for um parêntese fechado. Caso contrário, faz parte de um número.

Escreva uma função que pegue uma string contendo uma expressão matemática como seu único parâmetro e a divida em uma lista de tokens. Cada token deve ser um parêntese, um operador ou um número com um sinal opcional de + ou - (para simplificar, trabalharemos apenas com inteiros neste problema). Retorne a lista de tokens como o resultado da função.

Você pode presumir que a string passada para sua função sempre contém uma expressão matemática válida consistindo de parênteses, operadores e inteiros. Entretanto, sua função deve lidar com quantidades variáveis de espaços em branco entre esses elementos. Inclua um programa principal que demonstra sua função de tokenização lendo uma expressão do usuário e imprimindo a lista de tokens.

16. **Infix para posfix.** As expressões matemáticas são frequentemente escritas na forma de infixo, onde os operadores aparecem entre os operandos sobre os quais atuam. Embora seja uma forma comum, também é possível expressar expressões matemáticas na forma pós-fixada, em que o operador aparece depois de ambos os operandos. Por exemplo, a expressão infixa `3 + 4` é escrita como `3 4 +` na forma pós-fixada. Pode-se converter uma expressão infixada em uma forma pós-fixada usando o seguinte algoritmo:

Entrada: lista de tokens $infix$
Saída: lista de tokens $postfix$
crie uma nova lista vazia, *operadores*
crie uma nova lista vazia, $postfix$
para *cada token da lista* $infix$ **faça**
  se *token for um número inteiro* **então**
    Adicione o token ao final de $postfix$
  fim
  se *token for um operador* **então**
    enquanto *operadores não estiver vazio e o último item em operadores não é um parêntese aberto e precedência(token)<precedência(último item em operadores)* **faça**
      Remova o último item de *operadores* e adicione em $postfix$
    fim
    Adicione token ao final dos operadores
  fim
  se *token for um parêntese aberto* **então**
    Adicione o token ao final dos operadores
  fim
  se *token for um parêntese fechado* **então**
    enquanto *último item nos operadores não for um parêntese aberto* **faça**
      Remova o último item dos *operadores* e adicione-o ao $postfix$
    fim
    Remova de *operadores* o parêntese aberto
  fim
fim
enquanto *operadores não for uma lista vazia* **faça**
  Remova o último item dos *operadores* e adicione-o a $postfix$
fim
retorna *postfix*

Use a solução do exercício anterior para tokenizar uma expressão matemática. Em seguida, use o algoritmo acima para transformar a expressão da forma infixo para a forma pós-fixada. Seu código que implementa o algoritmo acima deve residir em uma função que recebe uma lista de tokens que representam uma expressão infixa como seu único parâmetro. Ele deve retornar uma lista de tokens que representam a expressão pós-fixada equivalente como seu único resultado. Inclua um programa principal que demonstra sua função infixo para pós-fixada

lendo uma expressão do usuário na forma infixo e exibindo-a na forma pós-fixada.

A finalidade da conversão da forma infixo para a forma pós-fixada ficará evidente quando você fizer o próximo exercício. Os exercícios 9 da lista 5 e 14 da lista atual podem ser úteis para concluir este problema.

Observação: os algoritmos implementados nos exercícios 16 e 17 não realizam nenhuma verificação de erro. Como resultado, você pode travar seu programa ou receber resultados incorretos se fornecer dados inválidos. Esses algoritmos podem ser estendidos para detectar entrada inválida e responder a ela de maneira apropriada. Isto fica como um exercício de estudo independente para o aluno interessado em aprimorar ainda mais suas habilidades de programação.

17. **Avaliação de expressão pós-fixada.** Avaliar uma expressão pós-fixada é mais fácil do que avaliar uma expressão infixa porque ela não contém nenhum colchete e não há regras de precedência de operador a serem consideradas. Uma expressão pós-fixada pode ser avaliada usando o seguinte algoritmo:

```
Crie uma nova lista vazia, valores
para cada token da expressão postfix faça
    se token for um número então
        Converta-o para um inteiro e adicione-o ao final de valores
    senão
        Remova um item do final dos valores e chame-o de direita
        Remova um item do final dos valores e chame-o de esquerda
        Aplique o operador aos itens esquerda e direita
        Anexe o resultado ao final de valores
    fim
fim
retorna o primeiro item de valores como o valor da expressão
```

Escreva um programa que leia uma expressão matemática em forma de infixo do usuário, a avalie e exiba seu valor. Usa suas soluções dos exercícios 15 e 16 junto com o algoritmo mostrado acima para resolver este problema.