Universidade Federal do Espírito Santo

Departamento de Computação e Eletrônica - CEUNES
Programação I / Programação Funcional
Prof. Oberlan Romão

Aula Prática 8 – Listas

Como visto na aula teórica, listas são um tipo de variável que permite o armazenamento de vários valores, que podem ser acessados por um índice. Listas são mutáveis, o que significa que seu conteúdo pode ser alterado durante a execução do programa, ou seja, itens podem ser adicionados ou removidos. Uma lista pode conter zero ou mais elementos de um mesmo tipo (duplicados ou não) ou de tipos diversos, podendo inclusive conter outras listas. O tamanho de uma lista é igual à quantidade de elementos que ela contém. Veja um exemplo de uma lista com quatro notas:

```
1 notas = [10.0, 8, 7.5, 5.5]
```

Para "descobrir" o número de elementos de uma lista, ou seja, o seu tamanho, podemos usar a função integrada `len`.

```
1 numNotas = len(notas)
2 print(f"Tamanho da lista: {numNotas}")
```

Além disso, é muito comum e muitas vezes necessário acessar um ou mais elemento da lista. Por exemplo, podemos estar interessado no terceiro valor da lista. Para isso, usamos o índice (ou seja, a posição) do elemento na lista. Cada elemento da lista possui um índice que especifica sua posição. O índice começa em 0, portanto, o índice do primeiro elemento é 0, o índice do segundo elemento é 1 e assim por diante. Também é possível usar índices negativos com listas para identificar as posições dos elementos em relação ao final da lista. O índice -1 identifica o último elemento em uma lista, -2 identifica o penúltimo elemento e assim por diante. Veja alguns exemplos:

```
1 print(notas[0])
2 print(notas[len(notas) - 1])
3 print(notas[-1])
4 print(-len(notas)]
```

Se precisamos percorrer a lista, podemos usar recursão para tal. Veja um exemplo de uma função que imprime os elementos de uma lista:

```
1 def imprime(L, i = 0):
2     if i < len(L):
3         print(L[i])
4         imprime(L, i + 1)
```

Pergunta: Na função foi feito `i < len(L)` (linha 2). Por que não foi usado o `i <= len(L)`? O que aconteceria se fizéssemos o `i <= len(L)`?

Exercício 1: (`ex1.py`) Modifique a função `imprime` para que a mesma só imprima os elementos de índices pares.
Exercício 2: (`ex2.py`) Baseada no código anterior, faça uma função `imprime` que recebe uma lista contendo apenas números inteiros e imprima apenas os número pares.

Como já dito, uma lista pode conter elementos de um mesmo tipo ou de tipos diversos. Por exemplo:

```
info = ["Pedro", 20, 1.73, "Vitoria", True, 9.5, 5]
```

Exercício 3: (`ex3.py`) Se usarmos a função `imprime` iremos imprimir todos os elementos da lista. Modifique a função para que a mesma imprima apenas os valores inteiros (do tipo `int`).

Analise a função abaixo, ela recebe uma lista e retorna a soma de seus elementos:

```
def soma(L, i = 0, s = 0):
    if i < len(L):
        return soma(L, i + 1, s + L[i])
    else:
        return s
```

Pergunta: Essa função pode ser usada para uma lista que contém valores de qual(is) tipo(s)?
Exercício 4: (`ex4.py`) Após aprender recursão, um programador resolveu usar a função `soma` para concatenar uma lista que contém apenas `string`.

- A função irá funcionar, ou seja, ela pode ser usada para concatenar `string`? Por quê?
- Se não funcionar, corrija a função.

Como uma lista pode aceitar diferentes tipos, é interessante saber se **todos** os elementos de uma lista são de um determinado tipo. Por exemplo, saber se todos os elementos são `int` ou `float` ou `str`, etc.
Exercício 5: (`ex5.py`) Faça uma função, chamada `todosNumeros`, que receba uma lista e retorna `True` se todos os elementos forem números (inteiros ou reais) e `False`, caso contrário.
Exercício 6: (`ex6.py`) Faça uma função recursiva que receba uma lista e um elemento `x` como parâmetros obrigatórios e retorne `True` se `x` está na lista e `False`, caso contrário.
Exercício 7: (`ex7.py`) Analise o código presente no arquivo `ex7.py` e, antes de executá-lo, tente descobrir o que o programa faz. Execute o programa e digite alguns valores. Deu tudo certo? Se não estiver, corrija-o e execute o programa. Como você deve ter percebido, o programa contabiliza o número de vezes que cada valor entre 0 e 10 foi digitado.

- Na linha 6 é feito `L = [0] * 11`, por que 11 e não 10?
- Ao invés de ficarmos digitando os valores, podemos usar um arquivo já com a entrada para o programa. Retire as mensagens dos `input`'s e comente a linha 22 (`print("Valor inválido!")`). Pelo terminal ou PowerShell, execute o seguinte comando: `python ex7.py < ex7.txt` (Linux) ou `cmd /c 'python ex7.py < ex7.txt'` (Windows). Esse comando usará/direcionará os dados de `ex7.txt` como entrada para o programa. Abra o arquivo `ex7.txt` para ver o conteúdo. Nesse arquivo, a primeira linha (que contém 100) será o valor atribuído à variável `n` (função `main`)

e cada um dos próximos 100 valores será atribuído à variável `valor` (função `contaValoresLidos`) em cada chamada recursiva da função.

- Também podemos direcionar a saída do programa para um arquivo específico. Ainda pelo terminal, execute o comando `python ex7.py < ex7.txt > saida.txt` (Linux) ou `cmd /c 'python ex7.py < ex7.txt > saida.txt'` (Windows). Note que agora nada é exibido no terminal. Toda saída é "jogada" para o arquivo `saida.txt`. Abra o arquivo `saida.txt` para ver o seu conteúdo.

- Volte com as mensagens dos `input`'s. Execute novamente o comando `python ex7.py < ex7.txt > saida.txt` (Linux) ou `cmd /c 'python ex7.py < ex7.txt > saida.txt'` (Windows). Abra o arquivo `saida.txt` e veja o resultado. O que apareceu nesse arquivo que não era "esperado"?

- Como você percebeu, todas as mensagens dos `input`'s foram direcionadas para o arquivo `saida.txt`. Por isso, nos miniEPs não usamos mensagem nos `input`'s, já que cada aluno poderia colocar mensagens diferentes e não haveria uma forma fácil de comparar a saída esperada com saída do programa;

- Retire novamente as mensagens dos `input`'s. Execute o comando `python ex7.py < ex7.txt > saida.txt` (Linux) ou `cmd /c 'python ex7.py < ex7.txt > saida.txt'` (Windows) . Se tudo deu certo, o arquivo `saida.txt` terá o seguinte conteúdo:

```
Foram lidos 97 valores validos
00: *
01: ****
02: ******
03: **********
04: ************
05: ****************
06: ********************
07: ***********
08: ******
09: ******
10: *****
```

Modifique o programa para que também seja impresso a quantidade lida de cada valor. A saída deve ser da seguinte forma:

```
Foram lidos 97 valores validos
00: [01] *
01: [04] ****
02: [06] ******
03: [10] **********
04: [12] ************
05: [16] ****************
06: [20] ********************
```

```
07: [11] ***********
08: [06] ******
09: [06] ******
10: [05] *****
```

- Abra o arquivo `saidaEsperada.txt` e compare com a saída do seu programa (presente no arquivo `saida.txt`). Seu programa gerou a saída esperada? Saber se a saída está idêntica a saída esperada não é uma tarefa fácil, mas podemos usar o computador para fazer essa comparação. Para isso, podemos usar o programa fc.exe (Windows) ou diff (Linux). No terminal, digite:
  - Windows: `fc.exe saida.txt saidaEsperada.txt`
  - Linux: `diff saida.txt saidaEsperada.txt`

  Se tudo deu certo, ou seja, se a saída do seu programa estiver idêntica a saída esperada, nada será exibido no terminal. Caso contrário, será mostrado a(s) diferença(s).

- Para testar os comandos anteriores, imprima um espaço a mais em algum `print` (por exemplo, imprima um espaço no fim da mensagem do `print` da função `main`), salve o arquivo e faça:
  `python ex7.py < ex7.txt > saida.txt && diff saida.txt saidaEsperada.txt` (Linux)
  `cmd /c 'python ex7.py < ex7.txt > saida.txt' && fc.exe saida.txt saidaEsperada.txt` (Windows)

  Note que com apenas uma única linha de código executamos o programa, geramos a saída e fazemos a comparação :)
  - Apareceu algo no terminal?
  - Faça outras modificações no arquivo `ex7.py` e veja como os comandos de comparação se comportam.

  Agora você já sabe como o Coffee faz para testar e saber se seu programa está correto!

Bom trabalho e divirta-se!