FIAP GRADUAÇÃO

COMPUTATIONAL THINKING FOR ENGINEERING

Fabio Henrique Pimentel
proffabio.pimentel@fiap.com.br

2024

Listas - criar

Listas são um tipo de variável que permite o armazenamento de vários valores, acessados por um índice.

Vejamos como criar uma lista:

```
L = []
```

Essa linha cria uma lista chamada L com zero elemento (list vazia). Os colchetes indicam que L é uma lista.

Vejamos como criar uma lista Z com 3 elementos:

```
Z = [15, 8, 9]
print(Z[0])
print(Z[1])
print(Z[2])
```

Podemos mudar o conteúdo de um elemento.

```
Z[0] = 7
print(Z[0])
print(Z)
```

Listas - criar

Vejamos um exemplo em que um aluno tem cinco notas e desejamos calcular a média aritmética:

```
notas = [6, 7, 5, 8, 9]
soma = 0
x = 0
while x < 5:
    soma += notas[x]
    x += 1
print(f"Média = {(soma/x):.2f}")
```

Obs.: a grande vantagem desse programa é que não precisamos declarar cinco variáveis para guardar as cinco notas. Todas as notas foram armazenadas em uma lista, utilizando um índice para identificar ou acessar cada valor.

Listas - criar

Vejamos uma modificação desse exemplo.
Vamos ler as notas uma a uma:

```
notas = [0, 0, 0, 0, 0]
soma = 0
x = 0
while x < 5:
    notas[x] = float(input(f"Nota {x+1} = "))
    soma += notas[x]
    x += 1
print(f"Média = {(soma/x):.2f}")
```

Exercício

1. Escreva um programa para ler 7 notas e calcular a média aritmética.

Listas - cópia e fatiamento

Quando fazemos:

```
L = [1, 2, 3, 4, 5]
V = L
print(L)
print(V)
V[0] = 6
print(V)
print(L)
```

Veja que, ao modificarmos V, modificamos também L. Isso porque uma lista em Python é um objeto e quando atribuímos um objeto a outro, estamos apenas copiando a mesma referência da lista. Nesse caso V funciona como um apelido de L, ou seja, V e L são a mesma lista.

Listas - cópia e fatiamento

Para criar um cópia independente de uma lista, vejamos a sintaxe:

```
L = [1, 2, 3, 4, 5]
V = L[:]
print(L)
print(V)
V[0] = 6
print(V)
print(L)
```

Listas - cópia e fatiamento

Podemos fatiar uma lista. Vejamos:

```
L = [1,2,3,4,5]
print(L[0:5])
print(L[:5])
print(L[:-1])
print(L[1:3])
print(L[1:4])
print(L[3:])
print(L[:3])
print(L[-1])
print(L[-2])
```

Um índice negativo começa a contar do último elemento, mas observe que começamos do -1. Assim L[0] representa o primeiro elemento, L[-1] o último, L[-2] o penúltimo e assim por diante.

Listas - tamanho

A função len retorna o número de elementos da lista. Vejamos:

```
L = [12, 9, 5]
print(len(L))
V = []
print(len(V))
```

A função len pode ser usada em repetições para controlar o limite dos índices:

```
L = [1, 2, 3]
x = 0
while x < len(L):
    print(L[x])
    x += 1
```

A vantagem é: se trocarmos os elementos de L o resto do programa continuará funcionando.

Listas - adição de elementos

Uma das principais vantagens das listas é poder adicionar novos elementos durante a execução do programa. Para adicionar um elemento ao fim da lista, utilizaremos o método append. Vejamos:

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
print(len(L))
```

Listas - adição de elementos

Vejamos um programa que lê números até que 0 (zero) seja digitado, depois o programa imprimirá na mesma ordem em que foram digitados:

```
L = []
while True:
    num = int(input("Digite um número (0 sai): "))
    if num == 0:
        break
    L.append(num)
x = 0
while x < len(L):
    print(L[x])
    x += 1
```

Listas - adição de elementos

Agora vamos ordenar a lista com os números digitados:

```
L = []
while True:
    num = int(input("Digite um número (0 sai): "))
    if num == 0:
        break
    L.append(num)
x = 0
while x < len(L):
    print(L[x])
    x += 1
print(L)
L.sort() #o método sort sem argumento ordena a lista em ordem ascendente
print(L)
L.sort(reverse=True) #o método sort com argumento reverse=True ordena a lista em ordem descendente
print(L)
```

Listas - adição de elementos

Outra maneira de adicionarmos elementos em uma lista é usando +

```
L = []
L = L + [1]
print(L)
L += [2]
print(L)
L += [3, 4, 5]
print(L)
```

Listas - adição de elementos

O método extend adiciona os elementos de uma lista a outra e o método append adiciona a lista dentro de outra lista.

```
L =[]
L.extend(["a", "b"])
print(L)
L.append(["c", "d"])
print(L)
print(len(L))
print(L[0])
print(L[1])
print(L[2])
```

Exercício

2. Faça um programa que leia duas listas e que gere uma terceira com os elementos das duas primeiras.

Listas - remoção de elementos

Para a remoção de elementos utilizaremos a instrução del. Vejamos:

```
L = ["a", "b", "c"]
print(L)
del L[1]
print(L)
```

Podemos também apagar fatias inteiras. Vejamos:

```
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(L)
del L[0:3]
print(L)
```

```
L.clear() #apaga a lista inteira
```

Listas - pesquisa

Podemos pesquisar se um elemento está ou não em uma lista. Vejamos o exemplo:

```
L = [15, 7, 27, 39]
pesquisa = int(input("Digite o valor a pesquisar: "))
x = 0
while x < len(L):
    if L[x] == pesquisa:
        achou = True
        break
    else:
        achou = False
    x += 1
if achou == True:
    print(f"{pesquisa} achado na posição {x}.")
else:
    print(f"{pesquisa} não encontrado.")
```

Exercício

3. Modifique o exemplo anterior de forma a realizar a mesma tarefa, mas sem utilizar a variável achou. Dica: observe a condição de saída do while.

FIAP GRADUAÇÃO