Linguagem de Programação Python

Listas e Tuplas

Professor: Ritomar Torquato

1

Listas e Tuplas

Objetivos: Concatenação de Listas; Geradores; Listas de Listas; Listas como Arrays

2

Concatenação de Listas

Adição como concatenação de listas.

```
>>> L = []
>>> L = L + [1]
>>> L
[1]
>>> L += [2]
>>> L
[1, 2]
>>> L += [3, 4, 5]
>>> L
[1, 2, 3, 4, 5]
```

```
>>> L = []
>>> L.extend([1])
>>> L
[1]
>>> L.extend([2])
>>> L
[1, 2]
>>> L.extend([3, 4, 5])
>>> L
[1, 2, 3, 4, 5]
```

Internamente o interpretador executa o método extend que adiciona os elementos de uma lista a outra.

3

Geradores

Guardar uma grande quantidade de dados em uma lista pode gerar um uso desnecessário da memória. Geradores permitem gerar uma lista dinamicamente invés de ocupar memória.

```
>>> anos = (ano for ano in range(1981, 1990))
>>> type(anos)
<class 'generator'>
```

4

Geradores

```
>>> for ano in anos:
        print(ano)


1981
1982
1983
1984
1985
1986
1987
1988
1989
```

```
>>> for ano in anos:
        print(ano)


>>> next(anos)
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    next(anos)
StopIteration
```

Um gerador precisa ser reiniciado após seu final. Usar um gerador já finalizado causa uma exceção **StopInteration**.

5

Geradores são úteis para criação de listas e tuplas dinamicamente.

```
>>> anos = tuple(ano for ano in range(1981, 1990))
>>> type(anos)
<class 'tuple'>
>>> anos
(1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989)
>>>
>>> anos = list(ano for ano in range(1981, 1990))
>>> type(anos)
<class 'list'>
>>> anos
[1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989]
>>>
>>> anos = [ano for ano in range(1981, 1990)]
>>> type(anos)
<class 'list'>
>>> anos
[1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989]
```

6

Listas de Listas

Adição de elementos do tipo Lista.

```
>>> L = ["a"]
>>> L.append(["b"])
>>> L.append(["c", "d"])
>>> L
['a', ['b'], ['c', 'd']]
>>> len(L)
3
```

```
>>> L[1]
['b']
>>> L[2]
['c', 'd']
>>> len(L[2])
2
>>> L[2][1]
'd'
```

Dessa forma é possível usar estruturas de dados como vetores, matrizes, árvores e registros.

7

Listas de Listas

São listas que possuem listas como seus elementos.

```
M = [
      [3, 8, 1, 5],
      [0, 2, 4, 7],
      [2, 5, 9, 3]
]
```

M =

3	8	1	5
0	2	4	7
2	5	9	3

```
print(M[0][0]) # 3
print(M[1][2]) # 4
print(M[2][0]) # 2
```

Necessitam de um índice para cada dimensão com elementos individuais.

8

Listas como Arrays

Array é como chamamos listas onde os **elementos simples** tem o mesmo tipo de dado (homogênea).

O array unidimensional é conhecido por **vetor**, enquanto o array bidimensional é denominado de **matriz**.

Vetor

V =

4	7	2	5	3

Matriz

M =

3	8	1	5
0	2	4	7
2	5	9	3

Array Tridimensional

T =

6	3	8	1
7	3	0	2	5	2
3	8	1	5	9	4	0	3
0	2	4	7	1	5
2	5	9	3

Arrays Multidimensionais podem ter quantas dimensões forem necessárias.

9

Listas como Arrays

A função `reverse()` retorna

```
>>> vetor = [9, 7, 8]
>>> vetor
[9, 7, 8]
>>> vetor[2]
8
>>> matriz = [vetor]
>>> matriz
[[9, 7, 8]]
>>> matriz[0][1]
7
>>> array = [matriz]
>>> array
[[[9, 7, 8]]]
>>> array[0][0][0]
9
```

vetor: Array Unidimensional
Lista simples
Um índice para acessar o elemento;

matriz: Array Bidimensional
Lista de vetores
Dois índices para acessar o elemento

array: Array Multidimensional
Lista de n Listas
3 ou mais índices para acessar o elemento

10

```
T = [
      [
        [3, 8, 1, 5],
        [0, 2, 4, 7],
        [2, 5, 9, 3]
      ],
      [
        [7, 3, 0, 2],
        [5, 0, 9, 4],
        [2, 4, 1, 5]
      ],
      [
        [6, 3, 8, 1],
        [2, 0, 5, 2],
        [1, 7, 0, 3]
      ]
]
```

```
print(T[0][0][0]) # 3
print(T[1][1][2]) # 9
print(T[2][0][0]) # 6
```

Necessitam de um índice para cada dimensão com elementos individuais.

T =

6	3	8	1
7	3	0	2	5	2
3	8	1	5	9	4	0	3
0	2	4	7	1	5
2	5	9	3

11

Preencher matriz

Uma matriz é uma lista com linhas do tipo vetor

```
from random import randint, seed
seed()

def preenche_matriz(linhas, colunas):
    matriz = [] # lista vazia
    for lin in range(linhas):
        linha = [] # cada linha é uma lista (vetor)
        for col in range(colunas):
            linha.append(randint(1, 50))

        # insere a linha na matriz
        matriz.append(linha)

    return matriz
```

Quantidade de linhas e colunas na matriz.

12

Percorrer matriz

É comum percorrer uma matriz pelos seus elementos ou pelos seus índices.

```
def imprime_matriz(matriz):
    for linha in matriz:
        print('|', end = '')
        for elemento in linha:
            print(f'{elemento:3d}', end = ' ')
        print('|')

def imprime_matriz_indice(matriz):
    for i_linha in range(len(matriz)):
        print('|', end = '')
        for i_coluna in range(len(matriz[i_linha])):
            print(f'{matriz[i_linha][i_coluna]:3d}', end = ' ')
        print('|')
```

13

Percorrer matriz

O resultado é o mesmo percorrendo pelos elementos ou pelos índices. Você pode escolher a melhor opção dependendo do problema que deseja resolver.

```
minha_matriz = preenche_matriz(4, 3)
minha_matriz[1][1] = 0
imprime_matriz(minha_matriz)
print()
imprime_matriz_indice(minha_matriz)
```

```
| 36   7  25 |
| 28   0  18 |
| 20  21  15 |
|  7  34  36 |
```

14

exemplo

Faça um programa que monte um array tridimensional 5 x 7 x 3, onde o conteúdo de cada elemento é igual a soma de seus índices.

15

```
def preenche_array(dim_0, dim_1, dim_2):
    dim_i = []
    for i in range(dim_0):
        dim_j = []
        for j in range(dim_1):
            dim_k = []
            for k in range(dim_2):
                dim_k.append(i+k+j)
            dim_j.append(dim_k)
        dim_i.append(dim_j)
    return dim_i


a = preenche_array(5, 7, 3)
for i in range(len(a)):
    imprime_matriz(a[i])
```

16

0

```
|  0   1   2 |
|  1   2   3 |
|  2   3   4 |
|  3   4   5 |
|  4   5   6 |
|  5   6   7 |
|  6   7   8 |
```

1

```
|  1   2   3 |
|  2   3   4 |
|  3   4   5 |
|  4   5   6 |
|  5   6   7 |
|  6   7   8 |
|  7   8   9 |
```

2

```
|  2   3   4 |
|  3   4   5 |
|  4   5   6 |
|  5   6   7 |
|  6   7   8 |
|  7   8   9 |
|  8   9  10 |
```

3

```
|  3   4   5 |
|  4   5   6 |
|  5   6   7 |
|  6   7   8 |
|  7   8   9 |
|  8   9  10 |
|  9  10  11 |
```

4

```
|  4   5   6 |
|  5   6   7 |
|  6   7   8 |
|  7   8   9 |
|  8   9  10 |
|  9  10  11 |
| 10  11  12 |
```

i x j x k

5 x 7 x 3

17