UNIVERSIDADE
BEIRA INTERIOR

Estruturas de Dados

Ferramentas e Aplicações em Biotecnologia

Vasco Ferrinho Lopes
vasco.lopes@ubi.pt

UBI
Ano letivo 2023-2024

Estruturas de Dados

Uma **estrutura de dados** compreende uma **agregação** de múltiplos **dados**, organizados de uma determinada forma: **linear/distribuída**, **homogénea/heterogénea.**

- Frequentemente, existe a necessidade de manipular conjuntos de dados de uma vez, como se fossem um só valor. Isto é, o processamento de um bloco de dados. Por exemplo, se quisermos representar os salários de um grupo de pessoas, ou os nomes dos meses do ano.

- Homogéneo significa que agrega dados do mesmo tipo

- Heterogénea significa que agrega dados de diferentes tipos.

Estruturas de Dados

Uma **Lista** é uma estrutura linear que pode ser homogénea ou heterógenea. Os valores contidos numa lista são armazenados contiguamente e podem ser **acedidos** através de um **índice**. Em Python, os valores de uma lista estão separados por virgula e delimitados por parênteses rectos.

```
[1, 2, 3, 1, 0, 7, 2] #------------> Lista de inteiros.
[1250.00, 2567.98, 112.56] #------> Lista de floats.
["Porto", "Covilhã", "Lisboa"] #--> Lista de strings.
```

Estruturas de Dados

Atribuição a variáveis:

```
salários = [1250.00, 2567.98, 112.56]
cidades  = ["Porto", "Covilhã", "Lisboa"]
```

Os documentos de uma lista podem ser acedidos individualmente, através do seu índice sequencial (inteiro), usando também os parênteses rectos:

```
>>> print(salários[0])
1250.0
>>> print(cidades[1])
Covilhã
>>> print(cidades[len(cidades)-1])
Lisboa
>>> print(cidades[-1])
Lisboa
>>> print(cidades[-2])
Covilhã
```

Estruturas de Dados

Podemos inclusive obter uma **sub-lista** usando o “slicing”, isto é, indicando um intervalo de índices, do género a:b, sendo a e b dois números inteiros, tais que a<b.

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

Estruturas de Dados

É ainda possível aplicar um conjunto de operações às listas, tais como:

```
>>> A = [4, 5, 6]
>>> B = [7, 8]
>>> print(A+B) #-----> concatenates two lists.
[4, 5, 6, 7, 8]

>>> A.append(9) #----> adds at the end.
>>> print(A)
[4, 5, 6, 9]

>>> A.insert(1,0) #--> insert 0 in position 1.
>>> print(A)
[4, 0, 5, 6, 9]

>>> C = [] #--------> creation of an empy list.
```

Exercício

Dado uma lista (abaixo mencionado): imprime todos os seus valores e a soma dos seus valores.

lista_valores = [1,2,5,10,15,20]

Estruturas de Dados

Um **conjunto** é uma estrutura **não linear** que agrega elementos não repetidos. Em Python, os valores de um conjunto estão separados por virgula e são delimitados pelas chavetas. **Não existe a noção de índice, tal como numa lista.** Aqui, um conjunto é conceptualmente equivalente à noção de um conjunto matemático

```
>>> A = {1,2,3,2}
>>> print(A)
{1, 2, 3}
>>> A.add(4)
>>> print(A)
{1, 2, 3, 4}

>>> 3 in A
True
```

```
>>> B = {3,4,5,7}
>>> print(A&B)
{3, 4}
>>> print(A|B)
{1, 2, 3, 4, 5, 7}
>>> {3,4} < B
True
>>> {1,2,3} == {2,1,3}
True
```

Estruturas de Dados

Uma lista pode ser facilmente transformado em um conjunto e vice-versa. Converter uma lista para um conjunto pode ser interessante em diversos casos, como quando se quer apenas os elementos únicos.

```
>>> L = [1,2,3,2,1,1,2,3,7]
>>> print(len(L))
9
>>> A = set(L)
>>> print(A)
{1, 2, 3, 7}
>>> print(len(A))
4
```

Exercício

Dado um conjunto (abaixo mencionado) imprima o valor máximo e minimo:

conjunto_valores = {1,2,5,10,15,20}

Estruturas de Dados

Um **tuplo** é uma estrutura linear sequencial e imutável com indexação igual à das listas. Em Python, os tuplos são delimitados por parênteses, sendo as suas componentes separadas por virgula. Um tuplo permite uma representação natural de um sistema de coordenadas, e.g., pontos ou vectores num espaço multidimensional.

```
>>> a = (1,0,-1)
>>> print(a)
(1,0,-1)
>>> a[0]
1
>>> len(a)
3
```

```
>>> list(a)
[1, 0, -1]
```

Estruturas de Dados

Uma **string** designa uma cadeia de caracteres, do tipo de dados **str**. É assim uma estrutura de dados **homogénea** (só caracteres) indexada sequencialmente, tal como numa lista ou num tuplo. O acesso a um caractere ou sequência de caracteres é realizado tal como numa lista ou tuplo, com uma combinação de parênteses rectos e ":".

```
>>> s = "01234567890123456789012345678901234567890"
>>> s = "Today is a nice day to study programming."
>>> print(s[0], s[-1], s[-2], len(s))
T . g 41
>>> s[:5]
Today
>>> print(s[23:-1], s[6:8], s[11:15])
study programming is nice
```

Estruturas de Dados

```
>>> s = 3 * "0123456789" #replicate
>>> print(s)
012345678901234567890123456789
>>> s = "universidade da beira interior"
>>> print(len(s))
30
>>> print(s.upper())
UNIVERSIDADE DA BEIRA INTERIOR
>>> print(s.title())
Universidade Da Beira Interior
>>> print(s.index("beira"))
16
>>> print(s.split())
['universidade', 'da', 'beira', 'interior']
>>> L = list("Estrela") #from str to list
>>> print(L)
['E', 's', 't', 'r', 'e', 'l', 'a']
```

Exercício

Dado duas strings (abaixo mencionado), indique se a segunda string é uma substring da primeira:

string_nome = “Biotecnologia”

string_dois = “tecn”

Estruturas de Dados

Dicionários são uma estrutura de dados associativa de elevada eficiência aquando a pesquisa. Nesta, uma chave está associada a um único valor. A sua estrutura geral é:

$$\{key_1 : value_1,\ key_2 : value_2, \ldots,\ key_n : value_n\}$$

Um exemplo da sua utilização é associar produtos ao seu preço:

```
produtos = {'cerveja':75, 'nata':65, 'cafe':55}
```

E existem diferentes operações que permitem lidar com esta estrutura de dados de forma mais fácil:

```
print(produtos.keys())
print(produtos.get('cafe'))
print(produtos['cafe'])
produtos['cafe'] = produtos['cafe'] + 1
```

Exercício

- Escreva um programa adicione uma nova chave a um dicionário.
  ex dicionário: dict_values = {1: “Olá”, 2: “Adeus”}

- Escreva um programa em python que dado um valor (“n”), cria um dicionário na forma (x, x*x) sendo que x=1,...,n. No fim, deve imprimir o dicionário resultante.