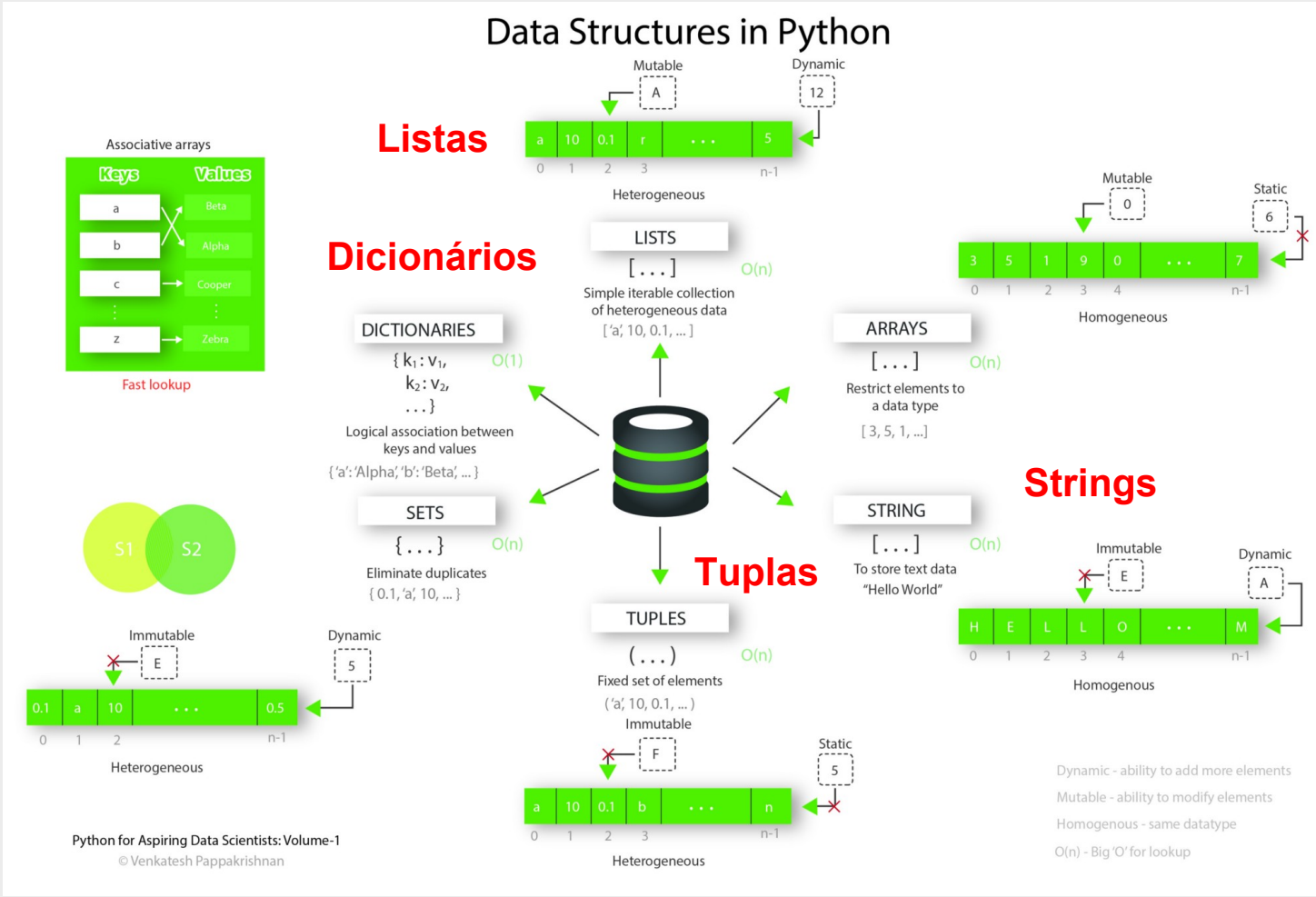
Algoritmos e Programação de Computadores

Tuplas e Dicionários

Ref.: material original (1o S., T. KLMN). por **Profa. Sandra Avila**, Instituto de Computação (IC/ Unicamp)

Agenda

- Estruturas de Dados em Python (II)
  - Tuplas
  - Dicionários
    - Operações
    - Métodos
  - Outras estruturas > listas (vista em Aula10), *strings*, …
- Exemplos


Data Structures in Python
Listas
Mutable
A
Dynamic
12
a
10
0.1
r
...
5
0
1
2
3
n-1
Heterogeneous
Associative arrays
Keys
Values
a
b
c
z
Beta
Alpha
Cooper
Zebra
Fast lookup
Dicionários
LISTS
[. . .]
O(n)
Simple iterable collection of heterogeneous data
['a', 10, 0.1, ...]
DICTIONARIES
{ k1 : v1,
k2 : v2,
. . . }
O(1)
Logical association between keys and values
{ 'a': 'Alpha', 'b': 'Beta', ... }
SETS
{ . . . }
O(n)
Eliminate duplicates
{ 0.1, 'a', 10, ... }
ARRAYS
[. . .]
O(n)
Restrict elements to a data type
[3, 5, 1, ...]
Mutable
0
Static
6
3
5
1
9
0
...
7
Homogeneous
Strings
STRING
[. . .]
O(n)
To store text data
"Hello World"
Immutable
E
Dynamic
A
H
E
L
L
O
...
M
Homogenous
S1
S2
Tuplas
TUPLES
(. . .)
O(n)
Fixed set of elements
('a', 10, 0.1, ...)
Immutable
Dynamic
E
5
0.1
a
10
...
0.5
Heterogeneous
Immutable
F
Static
5
a
10
0.1
b
...
n
Heterogeneous
Dynamic - ability to add more elements
Mutable - ability to modify elements
Homogenous - same datatype
O(n) - Big 'O' for lookup
Python for Aspiring Data Scientists: Volume-1
© Venkatesh Pappakrishnan

Estruturas de Dados

Sintaxe

- Lista > `[ "A", "C", "G","T" ]`
- Tupla > `( "A", "C", "G","T" )`
- Dicionário > `{ "A":"T", "C":"G", "G":"C","T":"A" }`

Nas tuplas, delimitadores “**(**“ e “**)**” são opcionais

Tuplas

- Estrutura semelhante a listas, porém, **tuplas são imutáveis**.
- Sintaxe: Tuplas são uma sequência (grupo ordenado) de elementos (items) separados por vírgulas, delimitados ou não por parênteses, isto é, os parênteses **não** são obrigatórios.
- Pode-se ainda misturar elementos de tipos diferentes.
- Porém, ao contrário de listas, as **tuplas são imutáveis**.
- Exemplo: `(18, "abril", 9.5, 1)` é uma tupla de 4 elementos.

Tuplas

- ( ) : tupla vazia.
- (33, 55, 77) : Uma tupla de números (int).
- (33, 3.3, (3+3j)): tupla de números mistos (combinação int, real-imag.).
- (33, “33”, [3,3]): Uma tupla de tipos mistos (combinação de tipos).
- `(("x","y"), ("X","Y"))` é uma tupla de 2 tuplas

- Pode-se pensar uma ***tupla*** como um pequeno *container*/pacote de informação

Tuplas

- Mais exemplos de tuplas.

```
tupla1 = ('abril', 18, 4, 2018)
tupla2 = (1, 2, 3, 4, 5, 6, 7)
tupla3 = "a", "b", "c", "d"
tupla4 = ("MC102", )
tupla5 = ()
```

- `tupla4` representa uma tupla com um único elemento. A vírgula após o elemento é necessária para diferenciar de uma expressão entre parênteses.

Tuplas

- O que será impresso?

```
t1 = 'A',
t2 = ('A')
print(type(t1))
print(type(t2))
```

```
<class 'tuple'>
<class 'str'>
```

Tuplas

- Como strings, tuplas são **imutáveis**.

```
a = (18, "abril", 9.5, 1)
a[2] = 9.0

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- A utilidade de uma lista imutável ficará mais clara na aula de hoje, na seção de dicionários.

Tuplas: Acessando Valores

- As operações para acessar os elementos ou sub-sequências de um lista e de uma string, também funcionam em tuplas.

```
a = (18, "abril", 9.5, 1)
a[2]
9.5    > i.e. a instrução print(a[2]) gera a saída 9.5   (*)

a[1:3]
("abril", 9.5)  > saida gerada pela instrução  print(a[1:3])  (*)
```

(*) Para visualizar corretamente a execução do código Python, deve-se incluir `print(tupla)`

Tuplas: Empacotamento e Desempacotamento

- Os elementos de uma tupla podem ser acessados de uma forma implícita na atribuição
  - Conhecido como desempacotamento: atribuir, individualmente, os elementos da tupla a outros objetos.
  - Um exemplo de desempacotamento de tupla:

```
x, y = (18, 20)

x
18    > i.e. a instrução Python print(x) gera a saída 18   (*)

y
20    > i.e. a instrução Python print(y) gera a saída 20   (*)
```

Tuplas: Empacotamento e Desempacotamento

- A tupla também pode ser implicitamente criada apenas separando os elementos por vírgula (conhecido como empacotamento).

```
18, 20
(18, 20)

"Setembro", 9.5
('Setembro', 9.5)
```

Em codificação Python : <codigo_fonte.py>
Ex.

```
tupla_0 = 18, 20
print(tupla_0)
```

➢ Execução retorna: `(18, 20)`

```
tupla_1 = "Setembro", 9.5
print(tupla_1)
```

➢ Execução retorna: `('Setembro', 9.5)`

Tuplas: Empacotamento e Desempacotamento

```
assassino, vitima, detetive = input().split()
```

```
Caio Estela Marcos
```

- O que será impresso?

```
assassino, vitima, detetive
```

```
('Caio', 'Estela', 'Marcos')
```

Dicionários

Dicionários

- Dicionários são estruturas de dados que associam uma chave com um valor.
- Os valores podem ser um dado de qualquer tipo, mas as chaves só podem ser dados de tipos imutáveis.
- As chaves precisam ser únicas.
- Um tipo dicionário é escrito da seguinte forma:

Um dicionário é denotado por { }.

```
dicionario = {chave1: valor1, ..., chaveN: valorN}
```

Dicionários

- Dicionários são estruturas de dados que associam uma chave com um valor.
- Os valores podem ser um dado de qualquer tipo, mas as chaves só podem ser dados de tipos imutáveis.
- As chaves precisam ser únicas.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
print(type(ra))
<class 'dict'>
```

Dicionários

- O dicionário abaixo pode representar os RAs de diversos alunos, com o nome (uma string, que é imutável) como chave e o valor associado a cada chave é o RA (um inteiro).
- Acessar o valor associado a uma chave é feito como no exemplo:

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
ra["Liz"]
229874
ra["Sofia"]
199745
```

Dicionários

- O valor associado a uma chave pode ser modificado, ou uma nova chave (e seu valor) podem ser incluídos no dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
ra
{'Hugo': 215793, 'Liz': 229874, 'Sofia': 199745}
```

Um dicionário é uma coleção não ordenada de pares chave-valor.

Dicionários

- O valor associado a uma chave pode ser modificado, ou uma nova chave (e seu valor) podem ser incluídos no dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
ra
{'Hugo': 215793, 'Liz': 229874, 'Sofia': 199745}
ra['Hugo'] = 215739
ra['Diego'] = 193278
ra
{'Diego': 193278, 'Hugo': 215739, 'Liz': 229874,
'Sofia': 199745,}
```

Operações em Dicionários

- O laço **for** aplicado a um dicionário faz a variável do laço passar por todas as **chaves** do dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
for x in ra:
        print(x)

Liz
Hugo
Sofia
```

Operações em Dicionários

- O operador **in** verifica se uma **chave** está no dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
"Sofia" in ra
True
"Aline" in ra
False
```

Operações em Dicionários

- Acessar uma chave que não existe causa erro de execução.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
ra['José']

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'José'
```

Métodos em Dicionários

- `items()` retorna todos os pares chave/conteúdo do dicionário.
- `keys()` retorna todas as chaves do dicionário.
- `values()` retorna todos os valores do dicionário.

Métodos em Dicionários

- `items()` retorna todos os pares chave/conteúdo do dicionário.
- `keys()` retorna todas as chaves do dicionário.
- `values()` retorna todos os valores do dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
ra.items()
dict_items([('Liz', 229874), ('Hugo', 215793), ('Sofia',
199745)])
ra.keys()
dict_keys(['Liz', 'Hugo', 'Sofia'])
ra.values()
dict_values([229874, 215793, 199745])
```

Métodos em Dicionários

- `items()` retorna todos os pares chave/conteúdo do dicionário.
- `keys()` retorna todas as chaves do dicionário.
- `values()` retorna todos os valores do dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
list(ra.items())
[('Liz', 229874), ('Hugo', 215793), ('Sofia', 199745)]
list(ra.keys())
['Liz', 'Hugo', 'Sofia']
list(ra.values())
[229874, 215793, 199745]
```

Métodos em Dicionários

- O método `get(chave)` retorna o valor atribuído à chave.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}

print(ra.get("Hugo"))
215793

print(ra.get("Maria"))
None

print(ra.get("Maria","N/A"))
N/A
```

Iterando em Dicionários

- Ao fazer uma iteração sobre dicionários, a chave e o valor correspondente podem ser recuperados ao mesmo tempo usando o método `items()`:

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}

for nome, numero in ra.items():
    print(nome, numero, sep=' ')

Liz 229874
Hugo 215793
Sofia 199745
```

Exemplo

Contando Letras

- Faça uma função que dada uma string, retorna a letra mais comum nessa string (em caso de empate retorne qualquer uma das mais frequentes).

Contando Letras

- Faça uma função que dada uma string, retorna a letra mais comum nessa string (em caso de empate retorne qualquer uma das mais frequentes).
  - Ideia: usar um dicionário para contar cada letra.
  - A letra é a chave do dicionário, e o valor será quantas vezes a letra foi encontrada.

Contando Letras

```
string = input("Digite uma string: ")
conta = {} # dicionário vazio
for letra in string:
    if letra in conta:
        conta[letra] += 1
    else:
        conta[letra] = 1 # adiciona letra no dicionário
letramais = ''
for key in conta:
      if not letramais: # nenhuma mais comum ainda
            letramais = key
      elif conta[key] > conta[letramais]:
            letramais = key
print(letramais)
```

Referências & Exercícios

- Os slides dessa aula foram baseados no material de MC102 dos Profs Sandra Ávila e Eduardo Xavier (IC/Unicamp)

- https://panda.ime.usp.br/pensepy/static/pensepy/11-Dicionarios/dicionarios.html