Tuplas e Dicionários

Vanessa Braganholo
vanessa@ic.uff.br

Tuplas

Tuplas

- Tuplas são sequências de valores, da mesma forma que listas
- Mas, existem diferenças...
  - Os valores de uma tupla, ao contrário de uma lista, são **imutáveis**
  - Tuplas usam parênteses enquanto listas usam colchetes

```
>>> lista = [1, 2, 3, 4]
>>> tupla = (1, 2, 3, 4)
```

Tuplas

- Tupla vazia

```
>>> tupla = ()
```

- Tupla com um único elemento (note a necessidade da vírgula, mesmo sendo um único elemento)

```
>>> tupla = (1,)
```

Acesso aos Elementos de uma Tupla

- Acesso é feito pela posição, da mesma forma que nas listas

```
>>> tupla = ("Maria", "Joao", "Carlos")
>>> tupla[0]
"Maria"
```

- Também é possível usar slices

```
>>> tupla = ("Maria", "Joao", "Carlos")
>>> tupla[0:2]
("Maria", "Joao")
```

Atualização de Tuplas

- Como são imutáveis, não é permitido atualizar os valores dentro de uma tupla

```
>>> tupla = ("Maria", "Joao", "Carlos")
>>> tupla[0] = "Ana"
TypeError: 'tuple' object does not support
item assignment
```

Operadores Básicos sobre Tuplas

Expressão	Resultado	Descrição
len((1,2,3))	3	Número de elementos que a tupla contém
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenação
(1,) * 4	(1,1,1,1)	Repetição
3 in (1, 2, 3)	True	Pertencimento
for x in (1,2,3): print(x)	1 2 3	Iteração

Dicionários

Agenda

- Como organizar os nomes e telefones dos seus amigos em Python?

Opção 1: usar uma lista

- Lista contém nome seguido de um ou mais telefones

```
>>> listaNomeTels = ["Maria", [99887766,99887755],
"Pedro", [92345678], "Joaquim", [99887711, 99665533]]
```

Opção 1: usar uma lista

- Lista contém nome seguido de um ou mais telefones

```
>>> listaNomeTels = ["Maria", [99887766,99887755],
"Pedro", [92345678], "Joaquim", [99887711, 99665533]]
```

- Como recuperar o telefone de Maria?

Opção 1: usar uma lista

- Lista contém nome seguido de um ou mais telefones

```
>>> listaNomeTels = ["Maria", [99887766,99887755],
"Pedro", [92345678], "Joaquim", [99887711, 99665533]]
```

- Como recuperar o telefone de Maria?

```
>>> tel = listaNomeTels[listaNomeTels.index("Maria")+1]
>>> tel
[99887766,99887755]
```

Alterações?

Remover Contato	• Exige remover dois elementos da lista...
Remover Telefone	• Exige remover um elemento de uma lista que está armazenada dentro de outra...
Acrescentar Telefone	• Exige acrescentar um elemento em uma lista que está armazenada dentro de outra...

Opção 2: usar duas listas

- Uma lista com os nomes
- Uma segunda lista com os telefones
- Correspondência pelas posições

```
>>> listaNomes = ["Maria", "Pedro", "Joaquim"]
>>> listaTelefones = [[99887766,99887755],
[92345678],  [99887711, 99665533]]
```

Opção 2: usar duas listas

- Uma lista com os nomes
- Uma segunda lista com os telefones
- Correspondência pelas posições

```
>>> listaNomes = ["Maria", "Pedro", "Joaquim"]
>>> listaTelefones = [[99887766,99887755],
[92345678],  [99887711, 99665533]]
```

- Como recuperar o telefone de Maria?

```
>>> tel = listaTelefones[listaNomes.index("Maria")]
>>> tel
[99887766,99887755]
```

Alterações?

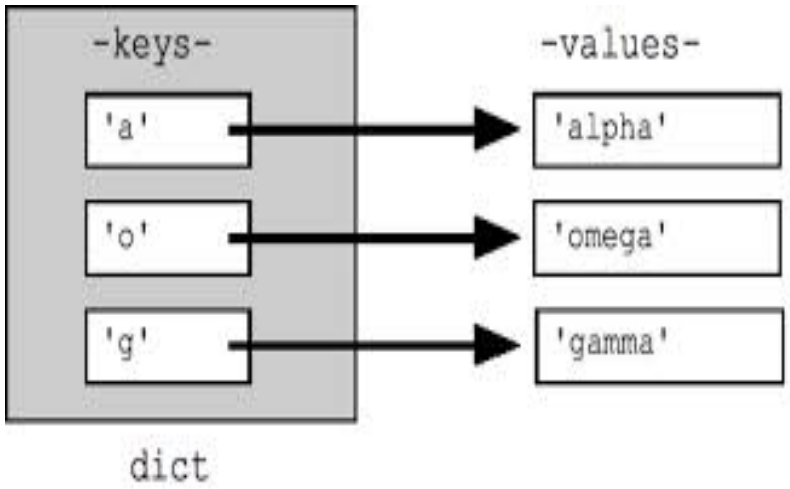
Remover Contato	• Exige remover elementos das duas listas...
Remover Telefone	• Exige remover um elemento de uma lista que está indexada por outra...
Acrescentar Telefone	• Exige acrescentar um elemento em uma lista que está indexada por outra...

Resumindo:

- Usando listas, a única forma de **indexação é usando números inteiros** (posição na lista)
- Isso sempre exige uma busca na lista auxiliar para encontrar a posição a ser usada para recuperar a informação desejada na segunda lista

Alternativa: Dicionário

- Estrutura de dados que implementa mapeamentos entre uma chave (*key*) e algum conteúdo (*value*)
  - Mapeamentos também são chamados de pares chave-valor
- A chave funciona como um **índice** para acessar o conteúdo
- Conteúdo pode ser qualquer coisa, inclusive outro dicionário



Voltando ao Exemplo da Agenda

- Qual dado deve servir como chave?
  - Por qual elemento quero fazer o acesso?
- Qual dado deve servir como conteúdo?
  - Qual(is) valor(es) quero associar à chave?

Voltando ao Exemplo da Agenda

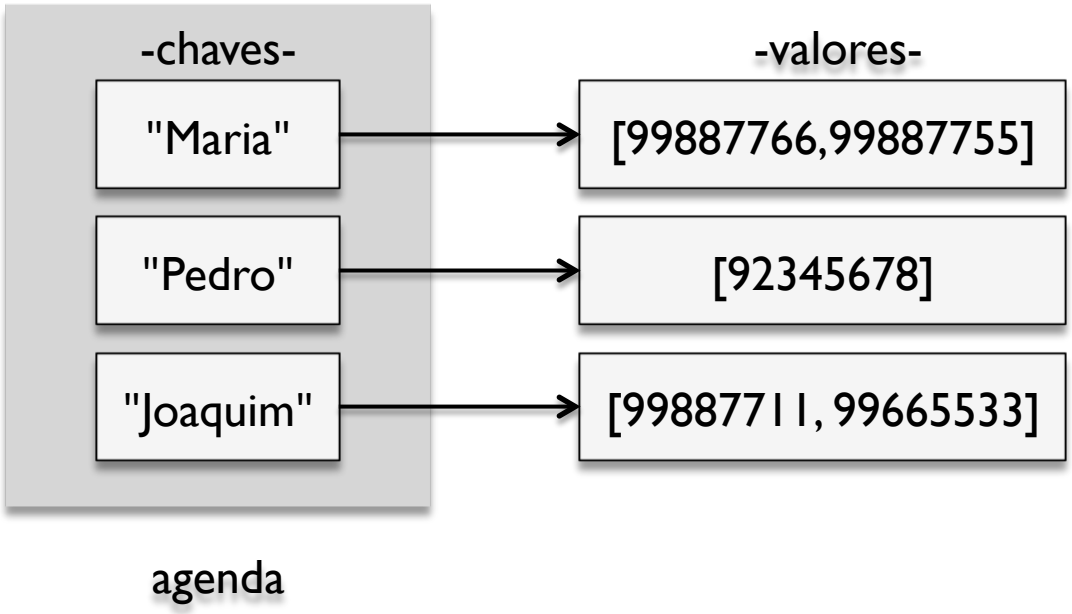
- Qual dado deve servir como chave?
  - Por qual elemento quero fazer o acesso?
- Qual dado deve servir como conteúdo?
  - Qual(is) valor(es) quero associar à chave?

nome

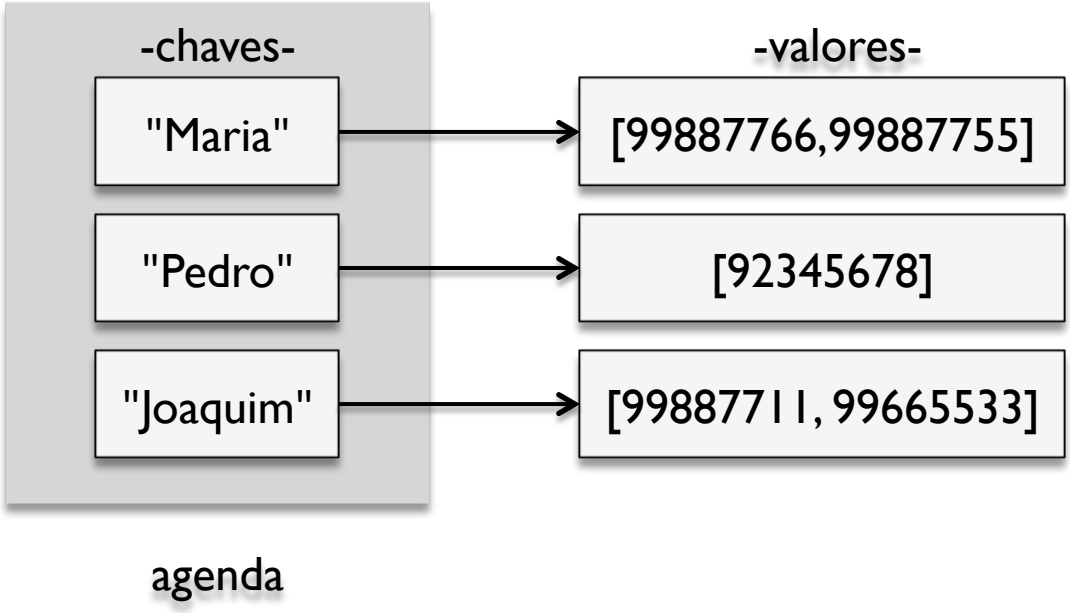
telefone

Agenda como um Dicionário

- Dicionário onde a chave é o **nome** e o conteúdo é a **lista de telefones**

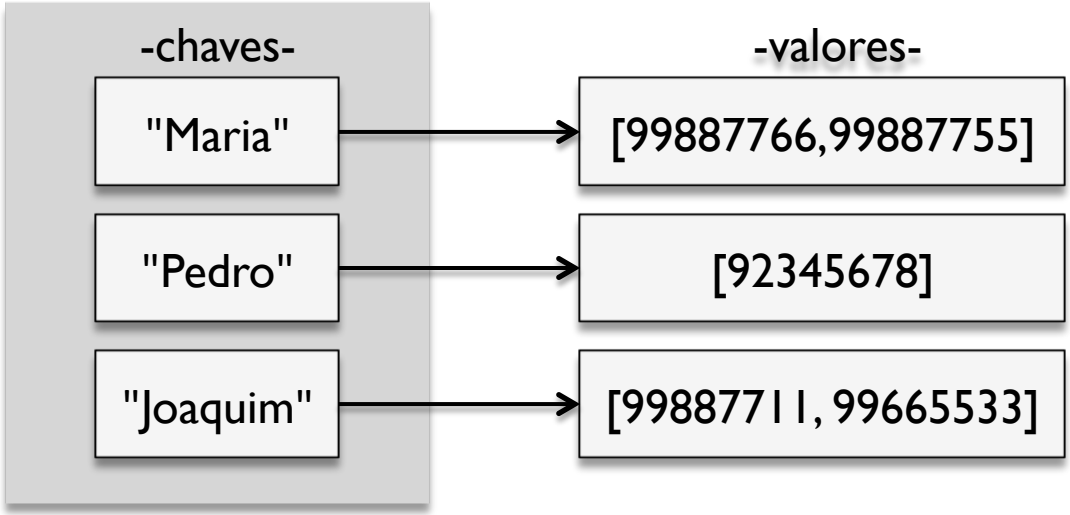


Agenda como um Dicionário




```
>>> agenda = {"Maria": [99887766, 99887755],
"Pedro": [92345678], "Joaquim": [99887711,
99665533]}
```

Agenda como um Dicionário



chave

```
>>> agenda = {"Maria": [99887766, 99887755],
"Pedro": [92345678], "Joaquim": [99887711,
99665533]}
```

valores

Organização de Dados num Dicionário

- Dicionário vazio:

```
>>> agenda = {}
```

- Chave é separada de seu valor por dois pontos

```
>>> agenda = {"Maria": [99887766, 99887755]}
```

- Cada par chave-valor é separado por vírgula dos pares seguintes

```
>>> agenda = {"Maria": [99887766, 99887755],
"Pedro": [92345678], "Joaquim": [99887711,
99665533]}
```

Acesso a Dados de um Dicionário

- Acesso é sempre feito pela chave

```
>>> agenda = {"Maria": [99887766, 99887755],
"Pedro": [92345678], "Joaquim": [99887711,
99665533]}
>>> telMaria = agenda["Maria"]
>>> telMaria
[99887766, 99887755]
```

Alterar valor de um conteúdo

- Usar a chave

```
>>> agenda = {"Maria": [99887766, 99887755],
"Pedro": [92345678], "Joaquim": [99887711,
99665533]}
>>> agenda["Pedro"] = [87654433]
>>> agenda
{"Maria": [99887766, 99887755], "Pedro":
[87654433], "Joaquim": [99887711, 99665533]}
```

Acrescentar Novos Valores

- Basta fazer atribuição a uma chave não existente
- Vale mesmo quando o dicionário está vazio

```
>>> agenda = {}
>>> agenda["Teresa"] = [65443322]
>>> agenda
{"Teresa": [65443322]}
```

Ordem

- As chaves dos dicionários não são armazenadas em nenhuma ordem específica
  - Na verdade, dicionários são implementados por tabelas de espalhamento (Hash Tables)
  - A falta de ordem é proposital

Dicionários x Listas

- Diferentemente de listas, atribuir a um elemento de um dicionário não requer que a posição exista previamente (isso ocorre porque não se trata de posição, e sim de valor da chave!)

```
>>> lista = []
>>> lista[10] = 5   # ERRO!

>>> dicionario = {}
>>> dicionario[10] = 5   # OK!
>>> dicionario
{10: 5}
```

Variável do tipo dicionários também armazena endereço de memória

- Com dicionários, ocorre o mesmo efeito que ocorre com cópia de listas – o que é copiado é o endereço de memória, e portanto, alterações nas cópias são refletidas umas nas outras

```
>>> d1 = {"Catarina":5}
>>> d2 = d1
>>> d1["Jonas"] = 20
>>> d2
{"Catarina": 5, "Jonas": 20}
```

copy()

- Retorna um outro dicionário com os mesmos pares chave/conteúdo

```
>>> d1 = {"Joao": 10, "Maria":20}
>>> d2 = d1.copy()
>>> d2["Pedro"] = 30
>>> d1["Joao"] = 40
>>> d1
{"Joao": 40, "Maria": 20}
>>> d2
{"Pedro": 30, "Joao": 10, "Maria": 20}
```

copy()

- Se conteúdo for lista, o que é copiado é apenas a referência...

```
>>> d1 = {"Joao":[1,2], "Maria":[3,4]}
>>> d2 = d1.copy()
>>> d2["Pedro"]=[5,6]
>>> d1["Joao"] += [3]
>>> d1
{"Joao": [1, 2, 3], "Maria": [3, 4]}
>>> d2
{"Pedro": [5, 6], "Joao": [1, 2, 3],
 "Maria": [3, 4]}
```

clear()

- Remove todos os elementos do dicionário

```
>>> idades = {"Joao":10, "Maria":12}
>>> idadesCriancas = idades
>>> idades.clear()
>>> idades
{}
>>> idadesCriancas
{}
```

Diferente de atribuir { } à variável

```
>>> idades = {"Joao":10, "Maria":12}
>>> idadesCriancas = idades
>>> idades = {}
>>> idades
{}
>>> idadesCriancas
{"Joao":10, "Maria":12}
```

Função dict()

- Função dict() pode ser usada para criar dicionários
- Pode receber dois tipos de parâmetros
  - Listas de tuplas, sendo que cada tupla contém uma chave e conteúdo
  - Sequencias de itens no formato **chave=valor**

dict() com lista de tuplas

- Cada tupla da lista contém uma chave e conteúdo

```
>>> produtos = dict([(10, 4.5), (20, 5.99)])
>>> valorProd = produtos[10]
>>> valorProd
4.5
>>> valorProd = produtos[20]
>>> valorProd
5.99
```

dict() com sequencia de itens chave=valor

- Sequencias de itens no formato **chave=valor**
- Nesse caso as chaves precisam ser **strings**, mas são escritas sem aspas

```
>>> produtos = dict(10=4.5,20= 5.99)
SyntaxError: keyword can't be an expression

>>> produtos = dict(prod10=4.5,prod20= 5.99)
>>> valorProd = produtos["prod10"]
>>> valorProd
4.5
```

fromkeys(lista, valor)

- Retorna um novo dicionário cujas chaves são os elementos de **lista** e cujos valores são todos iguais a **valor**
- Se **valor** não for especificado, o default é **None**

```
>>> {}.fromkeys([2,3])
{2: None, 3: None}
>>> dict.fromkeys(["Joao","Maria"],20)
{"Joao": 20, "Maria": 20}
```

get(chave, valor)

- Obtém o **conteúdo** associado à **chave**
- Se **chave** não existe, retorna **valor**
- Se **valor** não for especificado, chamadas de get para chaves inexistentes retornam **None**

```
>>> notas = {"Joao":[9.0,8.0], "Maria":
  [10.0]}
>>> notas.get("Maria")
[10.0]
>>> notas.get("Pedro")
None
>>> notas.get("Carlos","N/A")
N/A
```

in

- Retorna **True** se **chave** pertence ao dicionário e **False** caso contrário

```
>>> notas = {"Joao":[9.0,8.0], "Maria":
  [10.0]}
>>> "Pedro" in notas
False
```

items()

- Retorna uma lista com todos os pares chave/conteúdo do dicionário no formato de tupla

```
>>> notas = { "Joao":[9.0,8.0], "Maria":
[10.0] }
>>> notas.items()
[("Joao",[9.0,8.0]), ("Maria",[10.0])]
```

keys()

- Retorna uma lista com todas as chaves do dicionário

```
>>> notas = {"Joao":[9.0,8.0], "Maria":
[10.0]}
>>> notas.keys()
["Joao", "Maria"]
```

values()

- Retorna uma lista com todos os valores do dicionário

```
>>> notas = {"Joao":[9.0,8.0],
"Maria":[10.0]}
>>> notas.values()
[[9.0, 8.0] , [10.0]]
```

pop(chave)

- Obtém o valor correspondente à **chave** e remove o par chave/valor do dicionário

```
>>> notas = {"Joao":[9.0,8.0], "Maria":
  [10.0]}
>>> notas.pop("Joao")
[9.0,8.0]
>>> notas
{"Maria":[10.0]}
```

popitem()

- Retorna e remove um par chave/valor aleatório do dicionário
- Pode ser usado para iterar sobre todos os elementos do dicionário

```
>>> notas = {"Joao":[9.0,8.0],
  "Maria":[10.0]}
>>> notas.popitem()
{"Maria":[10.0]}
>>> notas
{”Joao":[9.0, 8.0]}
```

update(dic)

- Atualiza um dicionário com os elementos de outro
- Os itens em **dic** são adicionados um a um ao dicionário original (que foi usado para chamar a função update)
- É possível usar a mesma sintaxe da função **dict** para especificar **dic**

```
>>> x = {"a":1, "b":2, "c":3}
>>> y = {"z":9, "b":7}
>>> x.update(y)
>>> x
{"a":1, "c":3, "b":7, "z":9}
>>> x.update(a=7,c="xxx")
>>> x
{"a":7, "c":"xxx", "b":7, "z":9}
```

Iterando com Dicionários

- A iteração em elementos de um dicionário é feita a partir da chave
- Lembre-se de que com dicionários não temos ordem pré-definida

```
notas = {"Joao":[9.0,8.0], "Maria":[10.0]}
for nome in notas:
  media = sum(notas[nome])/len(notas[nome])
  print("A média de ", nome, " é: ", media)
```

Exercícios

1 – Escreva uma função que conta a quantidade de vogais em um texto e armazena tal quantidade em um dicionário, onde a chave é a vogal considerada.

2 – Escreva um programa que lê duas notas de vários alunos e armazena tais notas em um dicionário, onde a chave é o nome do aluno. A entrada de dados deve terminar quando for lida uma string vazia como nome. Escreva uma função que retorna a média do aluno, dado seu nome.

3 – Uma pista de Kart permite 10 voltas para cada um de 6 corredores. Escreva um programa que leia todos os tempos em segundos e os guarde em um dicionário, onde a chave é o nome do corredor. Ao final diga de quem foi a melhor volta da prova e em que volta; e ainda a classificação final em ordem (1o o campeão). O campeão é o que tem a menor média de tempos.

Exercícios

4 – Escreva um programa para armazenar uma agenda de telefones em um dicionário. Cada pessoa pode ter um ou mais telefones e a chave do dicionário é o nome da pessoa. Seu programa deve ter as seguintes funções:

- incluirNovoNome – essa função acrescenta um novo nome na agenda, com um ou mais telefones. Ela deve receber como argumentos o nome e os telefones.
- incluirTelefone – essa função acrescenta um telefone em um nome existente na agenda. Caso o nome não exista na agenda, você deve perguntar se a pessoa deseja incluí-lo. Caso a resposta seja afirmativa, use a função anterior para incluir o novo nome.
- excluirTelefone – essa função exclui um telefone de uma pessoa que já está na agenda. Se a pessoa tiver apenas um telefone, ela deve ser excluída da agenda.
- excluirNome – essa função exclui uma pessoa da agenda.
- consultarTelefone – essa função retorna os telefones de uma pessoa na agenda.

Créditos

- Slides de Aline Paes