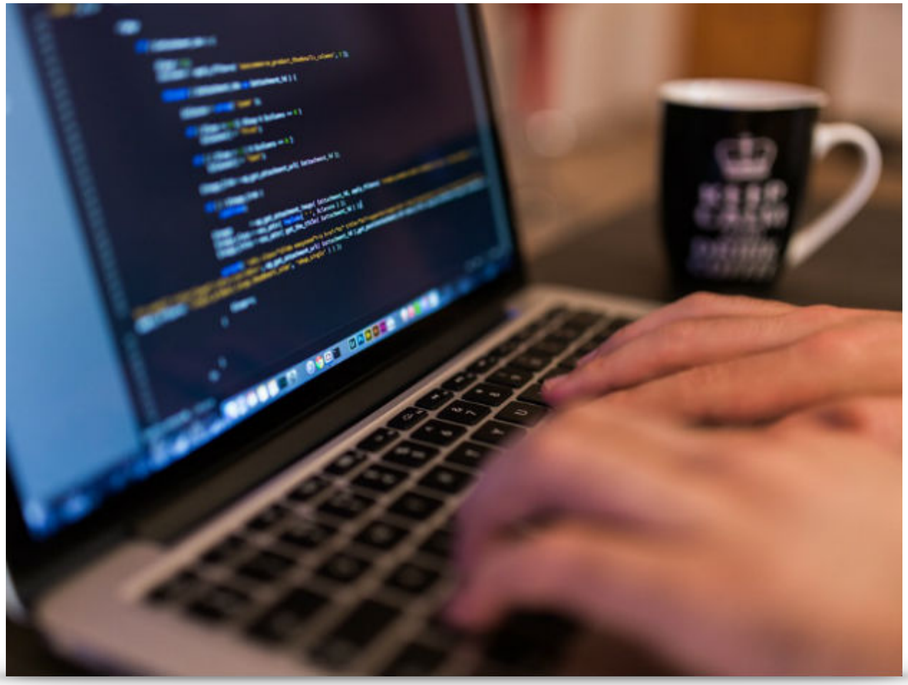
Profa. Dra. Raquel C. de Melo-Minardi
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais



MÓDULO 2 – PROGRAMAÇÃO
Dicionários

DICIONÁRIOS E HASHES

- Variáveis do tipo **dicionário** em **Python** são muito semelhantes a variáveis do tipo **hash** em **Perl**

- O que são **hashes**?

HASHES

- Uma variável do tipo ***hash*** representa um **conjunto não ordenado de pares chave-valor**
- Segundo [Ziviani, 2004], *hash* é um verbo no idioma inglês e significa
  - fazer picadinho (de carne ou vegetais para cozinhar)
  - fazer uma bagunça
- Nos **arranjos** (**sequências** em Python), os índices são números inteiros, crescentes, contíguos e são usados diretamente como *offsets* para endereçamento em memória
- Nos ***hashes***, as **chaves** são **quaisquer cadeias de caracteres** que devem obrigatoriamente ser transformadas em endereços de memória para serem usadas como chave

[Ziviani, 2004] Ziviani, Nivio. Projeto de algoritmos: com implementações em Pascal e C. Vol. 2. Thomson, 2004.

HASHES

- O processo de **transformação de chave** (cadeia de caracteres) consiste em dois processos:
  - computar uma **função de transformação**, que “pica” a cadeia de caracteres em pedaços e os utiliza para gerar um endereço em uma tabela (memória)
  - considerando que duas ou mais chaves podem ser transformadas em um mesmo endereço de uma tabela, implementar um método de **tratamento de colisões**

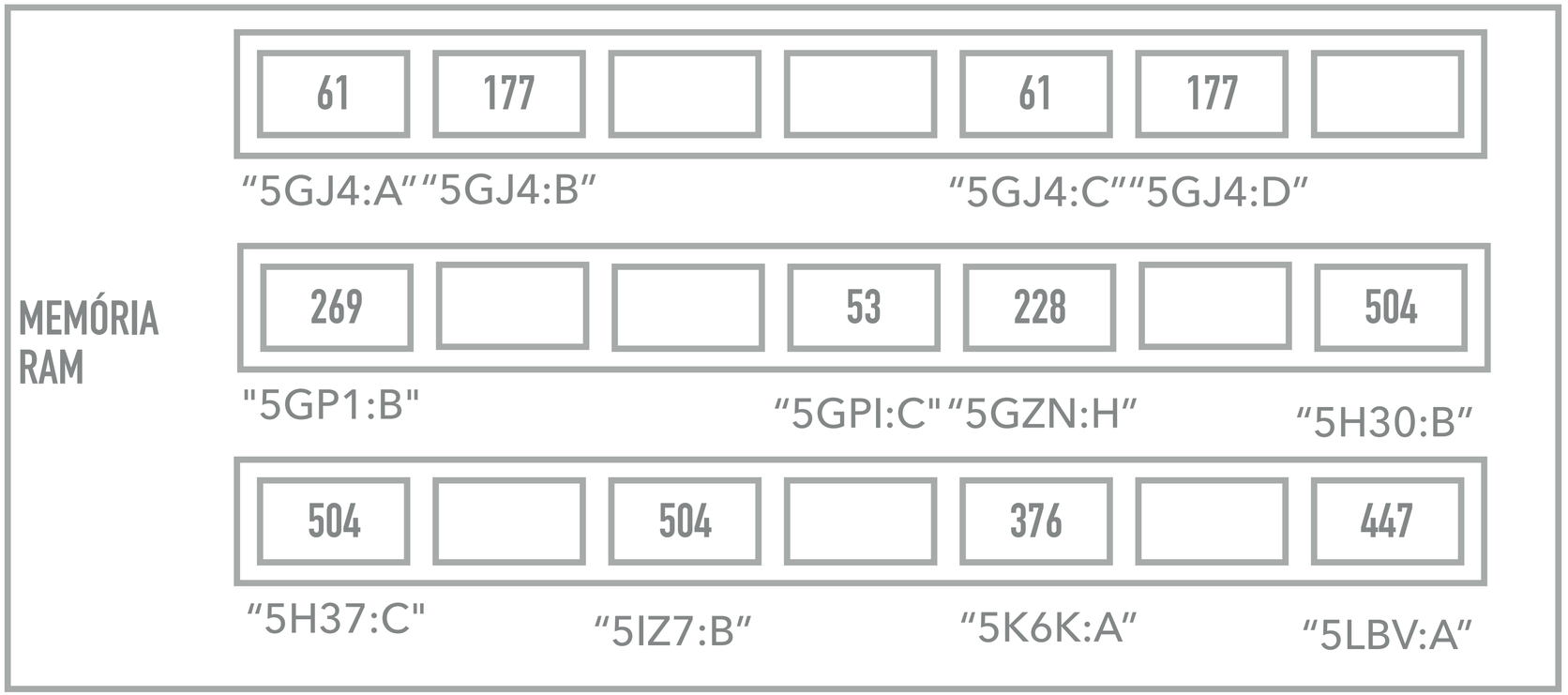
- Há diversos algoritmos para computar essas funções de transformação e tratar colisões e sua explicação está além do escopo desse curso

- O importante é explicar que o **Python já os implementa para o programador** e podemos usar variáveis do tipo *dicionário* de forma transparente a esses métodos

[Ziviani, 2004] Ziviani, Nivio. Projeto de algoritmos: com implementações em Pascal e C. Vol. 2. Thomson, 2004.

DICIONÁRIOS

- Diferentemente das variáveis do tipo sequência que ficam armazenadas em posições contíguas de memória, os **dicionários** podem ficar **espalhados em posições longínquas de memória** conforme a alocação pelo interpretador e sistema operacional



DICIONÁRIOS

Sequence Report **Total of 331 results.**

Back to Search Result Page

Click on column headers to sort up/down. Click again to reverse order. CSV

Page 1 of 17 View 1 - 20 of 331

	PDB ID	Chain ID	Entity ID	DB ID	DB Name	Sequence	Chain Len	Molecular	Entity Macı
	5GJ4	A	1	Q32ZE1	UniProt	1 GSHMTGKSVD MYIERAGDIT WEKDAEVTGN SPRLDVALDE 51 GPPMREKTGK R	61	67	
2	5GJ4	B	2	Q32ZE1	UniProt	1 SGALWDVPAP KEVKKGETTD GVYRVMTRRL LGSTQVGVGV 51 HVTKGAALRS GEGRLDPYWG DVKQDLVSYC GPWKLDAAWD 101 PPGERAKNIQ TLPGIFKTKD GDIGAVALDY PAGTSGSPIL 151 GNGVVIKNGS YVSAITQGKR EEETPVE	177	18980.50	Polypeptide
3	5GJ4	C	1	Q32ZE1	UniProt	1 GSHMTGKSVD MYIERAGDIT WEKDAEVTGN SPRLDVALDE	61	6721.37	Polypeptide

CHAVE ÚNICA

VALOR

CRIANDO E INICIALIZANDO DICIONÁRIOS

- Essas duas possibilidades apresentadas servem para
  - **inicializar** um dicionário com dados de interesse
  - **reinicializar** um dicionário que por ventura já possua valores
    - Note que o dicionário existente será **descartado** e conterá apenas os novos valores da inicialização

CRIANDO E INICIALIZANDO DICIONÁRIOS

- Em Python, um **dicionário** é representado como uma **sequência separada por vírgulas de pares chave-valor separados por “ : " e dentro de chaves** “ { } ”:

```
dic = {} # Inicializa ou reinicializa um dicionário existente
dic = {'5GJA:A':61, '5GJ4:B':177, '5GJ4:C':61, '5GJK:D':177, '5GP1:B':268}
```

ADICIONANDO ITENS A DICIONÁRIOS

- Uma outra outra possibilidade de inicialização, para posições individuais, tem a seguinte sintaxe:

```
dic['XXX'] = 999
```

  - Uma nova chave será criada no dicionário caso ainda não exista
  - O valor será atualizado caso já exista

ACESSANDO ITENS DE DICIONÁRIOS

- Itens de dicionários podem ser acessados diretamente

```
dic = {}
dic = {'5GJA:A':61, '5GJ4:B':177, '5GJ4:C':61, '5GJK:D':177, ‘5GP1:B':268}
```

```
print(dic['5GJA:A']) # Imprime 61
print(dic['5GJ4:B’]) # Imprime 177
print(dic['5GP1:B']) # Imprime 268
```

- O acesso a itens inexistentes retornará um erro

VERIFICAÇÃO DA EXISTÊNCIA DE ITENS EM DICIONÁRIOS

- Em alguns casos, é preciso verificar se um determinado valor está contido em um dicionário
  - Utilizamos o operador `in`, que indicará `True` se objeto pertencer ao conjunto, e `False` caso contrário:

```
print('5GJ4:C' in dic) # Imprime True
```

  - Adicionalmente ao operador `in`, podemos usar a versão complementar `not in`

COMPRIMENTO (OU TAMANHO) DE UM DICIONÁRIO

- Podemos obter o número de itens que compõem um dicionário através da função `len()` assim como nas outras estruturas de dados compostos em Python:

```
dic = {'5GJA:A':61, '5GJ4:B':177, '5GJ4:C':61, '5GJK:D':177, '5GP1:B':268}
len(dic) # Imprime 5
```

MESCLANDO DOIS DICIONÁRIOS

- É comum na manipulação de dicionários a necessidade de unir um conjunto de elementos a um outro conjunto, ou seja, a construção da união dos conjuntos
  - Para isso, usamos o método `update()`

```
dic = {'5GJA:A':61, '5GJ4:B':177, '5GJ4:C':61, '5GJK:D':177, '5GP1:B':268}
dic2 = {'1A6M:A':252, '2MM1:A':251}
dic.update(dic2)
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61, '5GJK:D': 177, '5GP1:B':
268, '1A6M:A': 252, '2MM1:A': 251}
```

VALORES MÍNIMOS E MÁXIMOS EM DICIONÁRIOS

- O Python oferece as funções `min()` e `max()` através das quais é possível encontrar, respectivamente, a menor e a maior chave de um dicionário
  - Funciona para cadeias de caracteres, obviamente, e faz uma ordenação lexicográfica

```
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61, '5GJK:D': 177, '5GP1:B':
268, '1A6M:A': 252, '2MM1:A': 251}
max(dic) # Retorna “1A6M:A”
min(dic) # Retorna “5GP1:B”
```

RETORNANDO E REMOVENDO VALORES DE DICIONÁRIOS

- O Python oferece o método `pop()` para **retornar** e **remover** itens de dicionários

```
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61, '5GJK:D': 177, '5GP1:B':
268, '1A6M:A': 252, '2MM1:A': 251}
dic.pop('1A6M:A') # Retorna 252
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61, '5GJK:D': 177, '5GP1:B':
268, '2MM1:A': 251}
```

RETORNANDO E REMOVENDO VALORES ALEATÓRIOS DE DICIONÁRIOS

- O Python oferece o método `popitem()` para retornar e remover itens **aleatórios** de dicionários

```
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61, '5GJK:D': 177, '5GP1:B': 268, '1A6M:A': 252, '2MM1:A': 251}
dic.popitem() # Retorna, por exemplo, ('5GP1:B', 268)
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61, '5GJK:D': 177, '1A6M:A': 252, '2MM1:A': 251}
```

- Pense em situações onde esse tipo de método poderia ser útil

REMOVENDO ITENS DE UM DICIONÁRIO PELO VALOR

- Há situações em que a chave dos elementos que se deseja remover é desconhecida e desejamos remover o item a partir do seu valor e para tal, em **listas**, usamos o método `remove(item)`
  - Este método **não funciona para dicionários**

COPIANDO DICIONÁRIOS

- Temos que ter cuidado com a cópia de dicionários em Python pois o comando:

```
dic2 = dic
```

- Apenas copia a referência do dicionário `dic` para `dic2`, ou seja, o nome `dic2` apontará para o mesmo endereço de memória para o qual aponta `dic`
  - Isso significa que quaisquer alterações feitas na estrutura / valores de `dic` se refletirá em `dic2` pois eles se tratam no mesmo objeto

COPIANDO DICIONÁRIOS

- Se desejamos criar um outro objeto que seja uma cópia real devemos usar o método `copy()`

```
dic2 = dic.copy()
```

- Nesse caso, **outro bloco de memória** (com **outro endereço**) será reservado para o novo dicionário `dic2` e todos **os pares chave-valor serão copiados** para essas novas posições de memória de forma que alterações em `dic` não refletirão em `dic2` e vice-versa

CONVERTENDO DICIONÁRIOS EM LISTA

- As vezes torna-se necessário a conversão entre estruturas diferentes, como exemplo, converter um **dicionário** em uma **lista**
- Há 3 métodos para extrair uma lista de um dicionário

```
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61, '5GJK:D': 177, '2MM1:A': 251}
```

  - `keys()` retorna a lista de chaves

```
# dict_keys(['5GJA:A', '5GJ4:B', '5GJ4:C', '5GJK:D', '2MM1:A'])
```

  - `values()` retorna a lista de valores

```
# dict_values([61, 177, 61, 177, 251])
```

  - `items()` retorna a lista de tuplas (pares chave-valor)

```
# dict_items([('5GJA:A', 61), ('5GJ4:B', 177),
('5GJ4:C', 61), ('5GJK:D', 177), ('2MM1:A', 251)])
```

CONVERTENDO DICIONÁRIOS EM LISTA

- Note os os **tipos dos objetos retornados não são listas propriamente ditas**
  - `keys()` retorna `dict_keys`
  - `values()` retorna `dict_values`
  - `items()` retorna `dict_items`
  - Para utilizar esses objetos como listas podendo, por exemplo, fazer a indexação direta, é interessante convertê-los para listas usando coerção de tipo (usando o *cast* `list`):

```
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61,
'5GJK:D': 177, '5GP1:B': 268, '1A6M:A': 252,
'2MM1:A': 251}
list(dic.keys())[0] # Retorna '5GJA:A'
```

CONVERTENDO LISTAS EM DICIONÁRIO

- Existe várias formas para que listas sejam convertidas em dicionários, no entanto, não podemos esquecer que um dicionário é um elemento que pares contém chave-valor

- Não podemos converter uma única lista em um dicionário sem chaves, até seria possível o oposto, porém, não há muita razão para este tipo de abordagem

  - É mais comum convertermos duas listas que contém a mesma quantidade de elementos em um dicionário

  - Assim, se nós tivermos em uma lista um conjunto de elementos que representem as chaves e noutra um conjunto de elementos que representem os valores, nós podemos converter essas listas em dicionário

  - O método `zip()` retorna uma lista contendo tuplas, onde o primeiro valor é o da primeira lista, e o segundo valor da tupla, corresponde a segunda lista

CONVERTENDO LISTAS EM DICIONÁRIO

- Com a função zip e a coerção de tipos, conseguimos obter um dicionário através de duas listas

- A seguir, nós criamos 2 listas e em seguida, convertemos as listas em uma lista de tuplas e depois, convertemos as listas em um dicionário:

```
aa3letras = ['ALA', 'CYS', 'ASP']
aa1letra = ['A', 'C', 'G']
list(zip(aa3letras, aa1letra))
# Retorna [('ALA', 'A'), ('CYS', 'C'), ('ASP', ‘G')]
dict(zip(aa3letras, aa1letra))
# Retorna {'ALA': 'A', 'CYS': 'C', 'ASP': 'G'}
```

ORDENANDO ITENS DE UM DICIONÁRIO

- **Não faz sentido ordenar um dicionário** visto que os elementos são armazenadas em posições não contíguas de memória
  - Podemos querer **imprimir** a lista de chaves e / ou valores de um dicionário em ordem alfabética ou de tamanho de chaves ou valores

```
# dic = {'5GJA:A': 61, '5GJ4:B': 177, '5GJ4:C': 61,
'5GJK:D': 177, '2MM1:A': 251}
lista = list(dic.keys())
lista.sort()
print(lista) # Imprime ['2MM1:A', '5GJ4:B', '5GJ4:C',
'5GJA:A', ‘5GJK:D']
```

DICIONÁRIOS OU LISTAS?

A SEGUIR...

- Vamos evoluir muito no tipo de tarefas que conseguimos executar após estudarmos as estruturas de repetição
  - Vamos tratar desse assunto nas aulas seguintes