Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

Programação de Computadores

Dicionários

Prof. José de Siqueira

2022/1

Organização de dados - Representação 1

- Consideremos o problema de contar quantas vogais tem um string `s`.
- Se queremos saber somente o total de vogais, basta acumular o resultado de `s.count(vogal)` para cada vogal.
- **Atenção!** Não confunda a variável `vogal` com o string `"vogal"`.
- No entanto, se queremos saber quantas vogais de cada tipo tem `s`, teremos que coletar a quantidade de cada uma delas em uma lista:

```
s = "o que e uai? uai e uai, uai!"

vogais = ["a", "e", "i", "o", "u"]

contagem  = [4,    3,   4,   1,   5]
```

Organização de dados - Representação 2

- Outra forma de organizar as vogais e suas contagens é fazer uma única lista em que cada vogal é seguida de sua contagem:

```
s = "o que e uai? uai e uai, uai!"

vogais = ["a", 4, "e", 3, "i", 4,  "o", 1,  "u", 5]
```

- Para ambas opções de organização dos dados, colocamos as seguintes perguntas:
  - Como imprimir a quantidade de cada vogal encontrada?
  - Como atualizar a lista de contagens para uma dada vogal?
  - Como incluir consoantes e suas contagens na lista de vogais, criando uma nova lista de letras?
  - Como retirar as vogais da lista de letras e criar uma lista só de consoantes?
  - Como alterar as contagens de determinadas letras?

Alterações

- Para remover letras ou contagem, é preciso alterar duas listas.
- Se as letras têm não uma só contagem, mas uma lista de contagens (para vogais com diacríticos, maiúsculas, ou seguidas de pontuação, por exemplo), para alterar os valores na lista de contagens é preciso acessá-la pelo índice da lista de vogais.

```
vogais = ["aáàã", "eéê", "ií", "oóôõ", "uú"]

contagens = [[4,3,1,2],[5,5,1],[3,1],[6,2,1,3],[4,1]]
```

- Tudo isso é factível, mas trabalhoso.
- Por isso Python propõe uma forma de organização de dados que facilita todas essas operações.

Dicionários

- Uma *lista* é uma coleção linear em que os elementos ficam em ordem (de inclusão).
- Um *dicionário* é uma coleção em que cada item tem uma etiqueta única e não tem ordem.
- Dicionários são a coleção de dados mais poderosa que o Python tem.
- Listas indexam seus elementos baseados na posição na lista.
- Dicionários são como uma sacola: cabe de tudo e não tem ordem.
- No entanto, indexamos as coisas que colocamos em um dicionário com uma **chave** (*key*).

Exemplo de dicionários

```
>>> sacola = dict()
>>> sacola['dinheiro'] = 27.50
>>> sacola['guloseimas'] = 4
>>> sacola['utilidades'] = 125
>>> print(sacola)
{'guloseimas': 4, 'utilidades': 125, 'dinheiro': 27.5}
>>> sacola['guloseimas'] = sacola['guloseimas'] + 5
>>> sacola['dinheiro'] = sacola['dinheiro'] - 5.50
>>> print(sacola)
{'guloseimas': 9, 'utilidades': 125, 'dinheiro': 22.0}
>>> vogais = dict()
>>> vogais['a'] = 0
>>> vogais['e'] = 0
>>> print(vogais)
{'a': 0, 'e': 0}
```

Dicionários são como listas, exceto que utilizam *chaves* em vez de índices para achar valores:

```
>>> l = list()
>>> l.append(11)
>>> l.append(17)
>>> print(l)
[11, 17]
>>> l[0] = 12
>>> print(l)
[12, 17]
```

Dicionários são como listas, exceto que utilizam *chaves* em vez de índices para achar valores:

```
>>> l = list()
>>> l.append(11)
>>> l.append(17)
>>> print(l)
[11, 17]
>>> l[0] = 12
>>> print(l)
[12, 17]
```

chave $= 0 \rightarrow$ valor $= 11$
chave $= 1 \rightarrow$ valor $= 17$

Comparação entre listas e dicionários

Dicionários são como listas, exceto que utilizam *chaves* em vez de índices para achar valores:

```
>>> l = list()
>>> l.append(11)
>>> l.append(17)
>>> print(l)
[11, 17]
>>> l[0] = 12
>>> print(l)
[12, 17]
```

```
>>> d = dict()
>>> d['idade'] = 24
>>> d['disciplina'] = 'DCC001'
>>> print(d)
{'idade': 24, 'disciplina': 'DCC001'}
>>> d['idade'] = 22
>>> print(d)
{'idade': 22, 'disciplina': 'DCC001'}
```

chave $= 0 \rightarrow$ valor $= 11$
chave $= 1 \rightarrow$ valor $= 17$

Comparação entre listas e dicionários

Dicionários são como listas, exceto que utilizam *chaves* em vez de índices para achar valores:

```
>>> l = list()
>>> l.append(11)
>>> l.append(17)
>>> print(l)
[11, 17]
>>> l[0] = 12
>>> print(l)
[12, 17]
```

chave = 0 → valor = 11
chave = 1 → valor = 17

```
>>> d = dict()
>>> d['idade'] = 24
>>> d['disciplina'] = 'DCC001'
>>> print(d)
{'idade': 24, 'disciplina': 'DCC001'}
>>> d['idade'] = 22
>>> print(d)
{'idade': 22, 'disciplina': 'DCC001'}
```

chave = 'idade' → valor=24
chave = 'disciplina' → valor='DCC001'

Qual flor mais aparece?

rosa	zínia	cravo	lírio	íris
cravo	dália	lírio	rosa	urze
zínia	rosa	íris	cravo	anis
lírio	íris	rosa	zínia	cravo
urze	dália	cravo	anis	lírio

Contando itens em um dicionário

- Uma das coisas mais comuns que queremos fazer com um dicionário é contar a frequência de seus itens.

```
>>> flores = dict()
>>> flores['rosa'] = 1
>>> flores['cravo'] = 1
>>> print(flores)
{'rosa': 1, 'cravo': 1}
>>> flores['rosa'] = flores['rosa'] + 1
>>> print(flores)
{'rosa': 2, 'cravo': 1}
```

Erro de execução (*traceback*) em dicionários

- Referenciar uma chave que não está no dicionário dá erro de execução (*traceback*).

```
>>> ccc = dict()
>>> print(ccc['orquídea'])
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
KeyError: 'orquídea'
```

- Para ver se uma chave existe em um dicionário, utilizamos o operador **in**.

```
>>> 'orquídea' in ccc
False
```

Quando encontramos uma nova flor

- Ao encontrarmos uma nova flor, colocamo-na no dicionário e iniciamos seu contador com 1.
- Se é a segunda vez ou mais que vemos a flor, recuperamos seu contador e o incrementamos de 1.

```
contadores = dict()
flores = ['rosa', 'zínia', 'cravo', 'lírio', 'íris']
for flor in flores :
  if flor not in contadores:
    contadores[flor] = 1
  else :
    contadores[flor] = contadores[flor] + 1
print(contadores)
```

```
{'cravo': 1, 'íris': 1, 'zínia': 1, 'lírio': 1, 'rosa': 1}
```

O método `get()` para dicionários

- Essas tarefas de verificar se uma chave (*key*) existe no dicionário e, se não existe, iniciá-la com um valor dado, é tão comum que Python nos oferece o método `get()` que faz isso para nós:

```
# Sem get()
if flor in contadores:
 x = contadores[flor]
else :
 x = 0
```

```
# Com get()
x = contadores.get(flor, 0)
```

- Valor inicial atribuído a `x` sem erro de execução.

Contagem simplificada com `get()`

- Podemos usar o método `get()` para iniciar uma chave que ainda não está no dicionário com um valor por omissão (*default*) e somente incrementar um a esse valor.

```
contadores = dict()
flores = ['rosa', 'zínia', 'cravo', 'lírio', 'íris']
for flor in flores :
  contadores[flor] = contadores.get(flor, 0) + 1
print(contadores)
```

```
{'rosa': 1, 'íris': 1, 'cravo': 1, 'zínia': 1, 'lírio': 1}
```

Padrão de contagem de palavras em texto com Python

- O padrão geral de contagem de palavras em uma linha de texto é separar (split) a linha em palavras e, então, percorrer as palavras e usar um dicionário para contar cada palavra individualmente.

```
conta = dict()
linha = input('Digite uma linha de texto: ')
palavras = linha.split()
print('Palavras: ', palavras)
print()
print('Contando...')
for palavra in palavras:
    conta[palavra] = conta.get(palavra,0) + 1
print()
print('Contagem: ', conta)
```

Contando palavras

```
$ python3.5 dicionario-de-palavras.py
Digite uma linha de texto: drome drome dromedario as areias do deserto
 sentem sono estou certo drome drome dromedario fecha os olhos o
 beduino fecha os olhos esta dormindo drome drome dromedario o frio da
 noite foi se embora fecha os olhos dorme agora drome drome dromedario
 dorme dorme a palmeira dorme dorme a noite inteira drome drome
 dromedario foi se embora o cansaco e voce dorme no meu braco drome
 drome dromedario drome drome dromedario drome drome dromedario

Palavras:  ['drome', 'drome', 'dromedario', 'as', 'areias', 'do',
 'deserto', 'sentem', 'sono', 'estou', 'certo', 'drome', 'drome',
 'dromedario', 'fecha', 'os', 'olhos', 'o', 'beduino', 'fecha', 'os',
 'olhos', 'esta', 'dormindo', 'drome', 'drome', 'dromedario', 'o',
 'frio', 'da', 'noite', 'foi', 'se', 'embora', 'fecha', 'os', 'olhos',
 'dorme', 'agora', 'drome', 'drome', 'dromedario', 'dorme', 'dorme',
 'a', 'palmeira', 'dorme', 'dorme', 'a', 'noite', 'inteira', 'drome',
 'drome', 'dromedario', 'foi', 'se', 'embora', 'o', 'cansaco', 'e',
 'voce', 'dorme', 'no', 'meu', 'braco', 'drome', 'drome', 'dromedario',
 'drome', 'drome', 'dromedario', 'drome', 'drome', 'dromedario']
Contando...
```

Contando palavras

```
Contagem:  'sono': 1, 'no': 1, 'as': 1, 'dromedario': 8,
 'se': 2, 'sentem': 1, 'drome': 16, 'a': 2, 'deserto': 1,
 'frio': 1, 'dormindo': 1, 'cansaco': 1, 'inteira': 1,
 'esta': 1, 'meu': 1, 'embora': 2, 'da': 1, 'noite': 2,
 'beduino': 1, 'os': 3, 'fecha': 3, 'estou': 1, 'voce': 1,
 'dorme': 6, 'foi': 2, 'do': 1, 'olhos': 3, 'braco': 1,
 'agora': 1, 'e': 1, 'palmeira': 1, 'certo': 1,
 'areias': 1, 'o': 3
```

- Apesar de dicionários não estarem em ordem, podemos usar um laço **for** para percorrer os itens de um dicionário.
- Na verdade, o laço **for** percorre todas as **chaves** do dicionário e, através da indexação de cada chave, obtemos os **valores**:

```
>>> contagens = { 'carlos' : 1 , 'fred' : 42, 'joao': 100}
>>> for chave in contagens:
...     print(chave, contagens[chave])
...
carlos 1
fred 42
joao 100
```

Obtendo listas de chaves e valores

- Podemos obter uma lista de chaves, ou de valores, ou de itens (tupla de chaves e valores) de um dicionário:

```
>>> jjj = { 'carlos' : 1 , 'fred' : 42, 'joao': 100}

>>> print(list(jjj))
['carlos', 'fred', 'joao']

>>> print(jjj.keys())
dict_keys(['carlos', 'fred', 'joao'])

>>> print(jjj.values())
dict_values([1, 42, 100])

>>> print(jjj.items())
dict_items([('carlos', 1), ('fred', 42), ('joao', 100)])
```

Iteração com o método `items()`

- Graças ao método `items()`, podemos iterar um dicionário com duas variáveis ao mesmo tempo, usando uma tupla (chave,valor) para desempacotar os itens do dicionário.

```
>>> jjj = { 'carlos' : 1 , 'fred' : 42, 'joao': 100}
>>> for (aaa,bbb) in jjj.items() :
...     print(aaa, bbb)
...
carlos 1
fred 42
joao 100
```

Lendo textos e palavras a partir de arquivos

- Uma outra maneira de entrar com dados em um programa é fornecer um arquivo para o programa ler os dados a partir dele.
- Para isso, precisamos abrir o arquivo físico e obter um **manipulador de arquivo** (*file handler*) para ele.
- Uma vez obtido o manipulador de arquivo, utilizamo-lo para ler e escrever do/no arquivo físico.

```
nome = input('Entre o nome do arquivo: ')
arquivo = open(nome)
texto = arquivo.read()
palavras = texto.split()
contagens = dict()
```

Contando a palavra mais frequente de um texto

```
for palavra in palavras:
    contagens[palavra] = contagens.get(palavra,0) + 1
contagemFrequente = None
palavraFrequente = None
for palavra,contagem in contagens.items():
    if (contagemFrequente is None
            or contagem > contagemFrequente):
        palavraFrequente = palavra
        contagemFrequente = contagem
print(palavraFrequente, contagemFrequente)
```

Calculando a média de notas em um dicionário

- Suponha que tenhamos um dicionário com nomes de alunos e listas de suas notas.
- Queremos imprimir as médias de notas de cada aluno, bem como a média da turma toda.

```
notas = {"Joao":[9.0,8.0], "Maria":[10.0]}
soma_das_medias = 0
for nome in notas:
    print(nome)
    media = sum(notas[nome])/len(notas[nome])
    soma_das_medias += media
    print("A média de ", nome, " é: ", media)
print("A média da turma é: ", soma_das_medias/len(notas))
```

Exercícios

- Nos exercícios abaixo, considere três representações para a contagem de vogais e consoantes e resolva os exercícios para as três representações.
  - **Representação 1**: uma tupla com duas listas distintas, a primeira com as vogais ou consoantes e a segunda com suas respectivas contagens: `([str],[int])`. Representação para o string vazio: `([],[])`.
  - **Representação 2**: uma única lista com vogais ou consoantes e suas respectivas contagens: `[str,int]`. Representação para o string vazio: `[]`.
  - **Representação 3**: uma única lista de tuplas com vogais ou consoantes e suas respectivas contagens: `[(str,int)]`. Representação para o string vazio: `[( )]`.

1. Faça uma função que receba um string e um inteiro de 1 a 3 e que retorne a representação do string de acordo com o inteiro.

Cabeçalho da função:	`exercicio_1(string,numero_da_representacao)`
Argumentos:	**string**: `str` **numero_da_representacao**: `int`

Saída:	**representacao**: `([str],[int])`
	`[str,int]`
	`[(str,int)]`

Observações:	Se o `numero_da_representacao` não for 1, 2 ou 3, retorna **`None`**. Senão, retorna a representação correspondente.

2. Faça uma função Python que retorne uma tupla: a contagem total de vogais e consoantes presentes em uma representação de string qualquer passada como argumento.

Cabeçalho da função:	`exercicio_2(representacao)`

Argumento:	**representacao**: `([str],[int])`
	`[str,int]`
	`[(str,int)]`

Saída:	**contagens_vogais_consoantes**: `(int,int)`
Observações:	A ordem das contagens na tupla retornada é `(vogais,consoantes)`, respectivamente. Se **representacao** for vazia, retorna **(0,0)**.

3. Faça funções Python para:
   3.1. retornar a tupla de listas de quantidade de vogais e consoantes, em ordem alfabética, para qualquer das 3 representações;

Cabeçalho da função:	`exercicio_3_1(representacao)`

Argumento:	**representacao**: `([str],[int])`
	`[str,int]`
	`[(str,int)]`

Saída:	**contagens_vogais_consoantes**: `([(str,int)],[(str,int)])`
Observações:	A ordem das contagens na tupla retornada é `(vogais,consoantes)`, respectivamente. As listas de vogais e consoantes devem ser ordenadas alfabeticamente.

3.2. dada uma letra, achar sua contagem para um string, para qualquer das 3 representações;

Cabeçalho da função:	`exercicio_3_2(representacao,letra)`

Argumento:	**representacao**: `([str],[int])`
	`[str,int]`
	`[(str,int)]`
	letra: `str`

Saída:	**int**: contagem da **letra** na **representacao**
Observação:	Se **letra** não estiver presente na **representacao**, retorna **None**.

3.3. retirar letras ou zerar contagens de qualquer representação de string;

Cabeçalho da função:	`exercicio_3_3(representacao,letra, código)`

Argumento:	**representacao**: `([str],[int])`
	`[str,int]`
	`[(str,int)]`
	letra: `str`
	código: `int`: 0 ou -1

Saída:	**representacao**
Observação:	Se **letra** não estiver presente na **representacao** ou, se esta for vazia, retorna **None**. **código** é **0**, para zerar a contagem da letra mantendo-a na **representacao** ou é -1, para retirar **letra** e sua contagem da **representacao**. Se **código** for diferente de 0 ou -1, retorna **None**.

4. Faça uma função (`exercicio_4_1`) que receba um string como parâmetro e retorne um dicionário com as frequências das letras no string. Se o string for vazio, retorna `None`. Refaça os exercícios 3.2 a 3.3 (`exercicio_4_2 a 4_3`, respectivamente) acima usando dicionários como representação única. As observações e retorno das funções são as mesmas indicadas nos exercícios 3.2 e 3.3.

Cabeçalhos das funções:

```
exercicio_4_1(string)
exercicio_4_2(dicionario,letra)
exercicio_4_3(dicionario,letra,codigo)
```

5. Crie uma agenda com, no mínimo, 26 nomes e 50 números de telefone, onde os nomes poderão ter 1 ou mais números de telefone (no máximo 4) associados a eles.
   - Os 26 nomes serão strings no seguinte formato: `'Abcde'`, `'Bcdef'`, `'Cdefg'`, ..., `'Xyzab'`, `'Yzabc'`, `'Zabcd'`, `'ABcde'`, `'BCdef'`, `'CDefg'`, ..., `'XYzab'`, `'YZAbc'`, `'ZAbcd'`, `'ABCde'`, `'BCDef'`, `'CDEfg'`, ... , `'XYZAb'`, `'YZAbc'`, `'ZABcd'`, `'ABCDe'`, `'BCDEf'`, `'CDEFg'`,....
   - Os 50 números serão strings no seguinte formato: `'123456789'`,`'234567890'`,`'345678901'`, `'456789012'`, ... , `'012345678'`, (inverte os 2 primeiros dígitos:) `'213456789'`, `'324567890'`,`'435678901'`,`'546789012'`,...,`'908765432'`, `'897654321'`, ..., (inverte os 3 primeiros dígitos:), `'321456789'`,`432567890'`, etc.

Exercícios

- Distribua os 26 nomes e os 50 números aleatoriamente na agenda, sendo que todos nomes têm que ter pelo menos um número e, no máximo, 4. Alguns números deverão ser atribuídos a mais de um nome.
- Pesquise sobre a biblioteca Python `random` (ver Referências) para escolher aleatoriamente um nome de uma lista, um número de 1 a 4 aleatoriamente e, de acordo com esse número aleatório, escolher tantos números de telefone de uma lista para cada um dos 26 nomes escolhidos aleatoriamente.

Faça funções para:

5.1. incluir uma nova entrada de nome ou telefone(s) na agenda:

Cabeçalho da função:	`exercicio_5_1(agenda,nome,numero)`
Argumentos:	**agenda**: `dict` **nome**,**numero**: `str`
Saída:	**dict**: agenda com o nome e número inseridos.
Observações:	Se nome ou número forem vazios, retorna o mesmo dicionário passado como argumento

5.2. apagar telefone(s) passados em uma lista ou um nome:

Cabeçalho da função:	`exercicio_5_2(agenda,nome,numero)`
Argumentos:	**agenda**: `dict` **nome**: `str` **numero**: `str`
Saída:	**dict**: agenda sem o nome ou o número passados como argumentos. Se nome ou número(s) não existirem, retorna **None**.
Observações:	Se **nome** for vazio, apaga o **numero** de todas entradas onde ele aparece; se **numero** for único para **nome**, apaga **nome** também. Se **numero** for vazio, apaga da agenda todos os números associados a **nome**, mas não apaga **nome**.

5.3. atualizar um nome ou telefone(s);

Cabeçalho da função:	`exercicio_5_3(agenda,nome1,numero1,nome2,\ numero2)`
Argumentos:	**agenda**: `dict` **nome1**,**numero1**,**nome2**, **numero2**: `str`
Saída:	**dict**: agenda ou **None**.
Observações:	1- **nome1!=”**,**numero1!=”**, **nome2!=”** e **numero2!=”**: troca nome e número; 2- **nome1!=”**,**numero1==”**, **nome2!=”** e **numero2==”**: troca nome; 3- **nome1!=”**,**numero1!=”**, **nome2==”** e **numero2!=”**: troca número; 4- **nome1!=”**,**numero1==”**, **nome2!=”** e **numero2!=”**: troca nome1 por nome2 e inclui número; apaga outros números existentes para **nome1**, se existirem. 5- **nome1==”**,**numero1!=”**, **nome2==”** e **numero2!=”**: troca número onde aparecer na agenda; 6- **nome1==”**,**numero1!=”**, **nome2!=”** e **numero2!=”**: inclui numero1 e numero2 para nome2; 7- senão, retorna **None**.

5.4. imprimir a agenda;

Cabeçalho da função:	`exercicio_5_4(agenda)`
Argumentos:	**agenda**: `dict`
Saída:	**str**: O string retornado, ao ser impresso, deverá imprimir um nome e seu(s) telefone(s) por linha: **nome1**: **numero1**; \n **nome2**: **numero2**; **numero3**; **numero4**; \n etc.

5.5. dado um nome, imprimir o(s) telefone(s) associado(s) ao nome.

Cabeçalho da função:	`exercicio_5_5(agenda,nome)`
Argumentos:	**agenda**: `dict` **nome**: `str`
Saída:	**str**: O string retornado, ao ser impresso, deverá imprimir o(s) telefone(s) associados ao **nome**.
Observação:	Se **nome** não constar da agenda, retorna **None**.

Referência

- https://docs.python.org/3/library/random.html