Introdução ao Processamento de Dados
Turma 3 (2020.1)

Dicionários e Arquivos

Gilson. A. O. P. Costa (IME/UERJ)

gilson.costa@ime.uerj.br

Tuplas

- São estruturas de dados parecidas com listas.
- Um valor do tipo tupla é definida como uma série de **valores separados por vírgulas e entre parênteses.**
- Na verdade, se não houver ambiguidade os parênteses são opcionais.

```
>>> a = [0, 1, 2, 3] # isto é uma lista
>>> b = (0, 1, 2, 3) # isto é uma tupla
>>> c = 4, 5, 6, 7  # isto é uma tupla
>>> c = 1,  # isto é uma tupla
```

Tuplas

- Outra diferença importante entre tuplas e listas é que não é possível modificar um elemento de uma tupla: ela é **imutável**.

```
>>> a = [0, 1, 2, 3] # isto é uma lista
>>> a[3] = 4 # isto é permitido
>>> print(a)
[0, 1, 2, 4]
```

Tuplas

- Outra diferença importante entre tuplas e listas é que não é possível modificar um elemento de uma tupla: ela é **imutável**.

```
>>> b = (0, 1, 2, 3) # isto é uma tupla
>>> print(b[2]) # acessando um elemento da tupla (isto pode)
2
>>> b[2] = 0 # isto não pode, vai dar erro!
```

Tuplas

- Tuplas são objetos sequencias e, como *strings* ou listas, **podem ser indexadas e fatiadas**.

```
>>> b = ('Maria', 'Engenharia Elétrica', 18)
>>> b[0]  # vai retornar uma string
'Maria'
>>> b[0:1]  # vai retornar uma tupla
('Maria',)
>>> b[0:2] # vai retornar uma tupla
('Maria', 'Engenharia Elétrica')
```

Tuplas

- A função ***list*** constrói uma lista a partir de uma sequência qualquer.
- A função ***tuple*** constrói uma tupla a partir de uma sequência qualquer.

```
>>> list("1234")
['1', '2', '3', '4']
>>> tuple("1234")
('1', '2', '3', '4')
>>> list((1,2,3))
[1, 2, 3]
>>> tuple([1,2,3])
(1, 2, 3)
```

Tuplas

Por que usar tuplas?

- Tuplas são mais rápidas do que listas.
- Se você está **definindo um conjunto constante de valores** e tudo o que vai fazer é iterar por este conjunto, use tupla em vez de lista.
- Torna o **código mais seguro**: você “protege contra gravação” os dados que não precisam ser alterados.
- Tuplas podem ser usadas como **chaves de dicionários** (por serem imutáveis).
- Listas não podem ser usadas como chaves de dicionário, porque as listas não são imutáveis.

Dicionários

- São estruturas de dados que implementam **mapeamentos**.
- Um mapeamento é uma coleção de **associações entre pares de valores**.
- O primeiro elemento do par é chamado de **chave** (*key*) e o outro de ***conteúdo*** (*value*).
- As chaves funcionam como índices, mas podem ser de qualquer tipo imutável: *int*, *float*, *str* e *tuple*.

{chave1: conteudo1, ..., chaveN: conteudoN}

Dicionários

Exemplos:

```
>>> copas = {"Brazil":5,"Alemanha":3,"Itália":3,"Argentina":1}
>>> copas
{'Brazil': 5, 'Alemanha': 3, 'Itália': 3, 'Argentina': 1}
>>> copas["Itália"]
3
>>> copas["Argentina"]
1
```

Dicionários

- **Atribuição** de um novo conteúdo/valor para uma **chave existente**:

```
>>> copas = {"Brasil":5,"Alemanha":3,"Itália":3,"Argentina":1}
>>> copas["Alemanha"] = 4
>>> print(copas)
{'Brasil': 5, 'Alemanha': 4, 'Itália': 3, 'Argentina': 1}
>>> copas["Alemanha"]
4
```

Dicionários

- **Atribuição** de um conteúdo/valor para uma **chave inexistente:**

```
>>> copas = {"Brasil":5,"Alemanha":4,"Itália":3,"Argentina":1}
>>> copas["França"] = 2
>>> print(copas)
{'Brasil': 5, 'Alemanha': 4, 'Itália': 3, 'Argentina': 1 , 'França': 2}
```

- Cria um novo par chave/conteúdo no dicionário!

Dicionários

- Perceba que você pode usar listas (ou tuplas) como **conteúdo**, de forma que uma chave pode estar associada a **diversos valores**.
- Pense num conjunto de endereços. A chave pode ser o nome da pessoa, e o conteúdo, uma lista com: endereço, número, complemento e CEP.

```
>>> endereco = {}
>>> endereco['João'] = ['Rua A', 23, 'Apto.101', '222808-100']
>>> endereco['Maria'] = ['Rua B', 4, 'Casa', '222500-050']
>>> print(endereco)
{'João': ['Rua A', 23, 'Apto.101', '222808-100'], 'Maria': ['Rua B', 4, 'Casa', '222500-050']}
```

Dicionários

- Perceba que você pode usar listas (ou tuplas) como **conteúdo**, de forma que uma chave pode estar associada a **diversos valores**.

```
>>> print(endereco)
{'João': ['Rua A', 23, 'Apto.101', '222808-100'], 'Maria': ['Rua B', 4, 'Casa', '222500-050']}
>>> endereco['João'][0]
'Rua A'
>>> endereco['Maria'][3]
'222500-050'
```

Dicionários

Exemplo: programa para calcular o índice de massa corporal (IMC) de uma pessoa com idade entre 10 e 20 anos.

$$\text{IMC} = \text{peso (em quilos)} \div \text{altura}^2 \text{ (em metros)}$$

Homens	Categoria		
Idade	Baixo Peso	Adequado	Sobrepeso
10	< 14,42	entre 14,42 e 19,60	>= 19,60
11	< 14,83	entre 14,83 e 20,35	>= 20,35
12	< 15,24	entre 15,24 e 21,12	>= 21,12
13	< 15,73	entre 15,73 e 21,93	>= 21,93
14	< 16,18	entre 16,18 e 22,77	>= 22,77
15	< 16,59	entre 16,59 e 23,63	>= 23,63
16	< 17,01	entre 17,01 e 24,45	>= 24,45
17	< 17,31	entre 17,31 e 25,28	>= 25,28
18	< 17,54	entre 17,54 e 25,95	>= 25,95
19	< 17,80	entre 17,80 e 26,36	>= 26,36

Mulheres	Categoria		
Idade	Baixo Peso	Adequado	Sobrepeso
10	< 14,23	entre 14,23 e 20,19	>= 20,19
11	< 14,60	entre 14,60 e 21,18	>= 21,18
12	< 14,98	entre 14,98 e 22,17	>= 22,17
13	< 15,36	entre 15,36 e 23,08	>= 23,08
14	< 15,67	entre 15,67 e 23,88	>= 23,88
15	< 16,01	entre 16,01 e 24,29	>= 24,29
16	< 16,37	entre 16,37 e 24,74	>= 24,74
17	< 16,59	entre 16,59 e 25,23	>= 25,23
18	< 16,71	entre 16,71 e 25,56	>= 25,56
19	< 16,87	entre 16,87 e 25,85	>= 25,85

Dicionários

Exemplo: programa para calcular o índice de massa corporal (IMC) de uma pessoa com idade entre 10 e 20 anos.

$$\text{IMC} = \text{peso (em quilos)} \div \text{altura}^2 \text{ (em metros)}$$

```
categorias = {"m": {10: [14.42, 19.60],  "f": {10: [14.23, 20.19],
                    11: [14.83, 20.35],         11: [14.60, 21.18],
                    12: [15.24, 21.12],         12: [14.98, 22.17],
                    13: [15.74, 21.93],         13: [15.36, 23.08],
                    14: [16.18, 22.77],         14: [15.67, 23.88],
                    15: [16.59, 23.63],         15: [16.01, 24.29],
                    16: [17.01, 24.45],         16: [16.37, 24.74],
                    17: [17.31, 25.28],         17: [16.59, 25.23],
                    18: [17.54, 25.95],         18: [16.71, 25.56],
                    19: [17.80, 26.36]},        19: [16.87, 25.85]}}
```

Dicionários


```
categorias = {
"m": {10: [14.42, 19.60], 11: [14.83, 20.35], 12: [15.24, 21.12], 13: [15.74, 21.93], 14: [16.18, 22.77], 15: [16.59, 23.63],
16: [17.01, 24.45], 17: [17.31, 25.28], 18: [17.54, 25.95], 19: [17.80, 26.36]},
"f": {10: [14.23, 20.19], 11: [14.60, 21.18], 12: [14.98, 22.17], 13: [15.36, 23.08], 14: [15.67, 23.88], 15: [16.01, 24.29],
16: [16.37, 24.74], 17: [16.59, 25.23], 18: [16.71, 25.56], 19: [16.87, 25.85]}
}
peso = float(input("Digite o seu peso em Kg: "))
altura = float(input("Digite sua altura em m: "))
idade = int(input("Digite sua idade: "))
sexo = input("Digite o seu sexo (m ou f): ")
imc = round(peso / (altura ** 2), 2)
print("Seu IMC é: ", imc)
if (10 <= idade < 20):
    if imc < categorias[sexo][idade][0]:
        print("Baixo Peso")
    elif imc < categorias[sexo][idade][1]:
        print("Adequado")
    else:
        print("Sobrepeso")
```

Função *dict*

- A função ***dict(...)*** pode ser usada para criar dicionários de duas formas.
- Usando como **parâmetros** uma **lista de tuplas**, cada uma com um par chave/conteúdo:

```
>>> idade = dict([('João',23), ('Maria',18), ('Leopoldina',77)])
>>> idade['Leopoldina'] + idade['João']
100
>>> print(idade)
{'João': 23, 'Maria': 18, 'Leopoldina': 77}
```

Função *dict*

- A função ***dict(...)*** pode ser usada para criar dicionários de duas formas.
- Uma sequência de itens no formato *chave=valor*.
- Nesse caso, as chaves têm que ser *strings*, mas são escritas sem aspas:

```
>>> idade = dict(João=23, Maria=18, Leopoldina=77)
>>> print(idade)
{'João': 23, 'Maria': 18, 'Leopoldina': 77}
>>> endereco = dict(João=['Rua A', 23, 'Apto.101', '222808-100'], Maria = ['Rua B', 4, 'Casa', '222500-050'])
>>> endereco['João'][3]
'222808-100'
```

Dicionários

- Perceba que como para listas/vetores e matrizes, quando você faz uma atribuição para todo o dicionário, você está apenas criando uma nova referência para ele.

```
>>> idade = dict(João=23, Maria=18, Leopoldina=77)
>>> age = idade
>>> print(age)
{'João': 23, 'Maria': 18, 'Leopoldina': 77}
>>> age['Maria'] = 19
>>> print(idade)
{'João': 23, 'Maria': 19, 'Leopoldina': 77}
```

Método *copy*

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

```
>>> idade = dict(João=23, Maria=18, Leopoldina=77)
>>> age = idade.copy()
>>> print(age)
{'João': 23, 'Maria': 18, 'Leopoldina': 77}
>>> age['Maria'] = 19
>>> print(age)
{'João': 23, 'Maria': 19, 'Leopoldina': 77} # idade de Maria mudou!
>>> print(idade)
{'João': 23, 'Maria': 18, 'Leopoldina': 77} # idade de Maria não mudou!
```

Método *copy*

- Retorna um outro dicionário com os mesmos pares chave/conteúdo.
- Se o conteúdo é uma lista, o problema persiste:

```
>>> endereco = dict(João=['Rua A', 23, 'Apto.101', '222808-100'], Maria = ['Rua B', 4, 'Casa', '222500-050'])
>>> address = endereco.copy()
>>> address['João'][1] = 32
>>> print(endereco['João'])
['Rua A', 32, 'Apto.101', '222808-100']
```

Cópia de Dicionários (*deepcopy*)

- Se o que se quer é uma cópia completa (e não referências para os mesmos dados), deve-se usar a função *deepcopy* do módulo *copy*.

```
>>> import copy
>>> endereco = dict(João=['Rua A', 23, 'Apto.101', '222808-100'], Maria = ['Rua B', 4, 'Casa', '222500-050'])
>>> address = copy.deepcopy(endereco)
>>> address['João'][1] = 32
>>> print(address['João'])
['Rua A', 32, 'Apto.101', '222808-100']
>>> print(endereco['João'])
['Rua A', 23, 'Apto.101', '222808-100']
```

Método *get*

- Obtém o conteúdo associado a uma chave.
- Não causa erro caso a chave não exista (retorna *None*).

```
>>> idade = dict(João=23, Maria=18, Leopoldina=77)
>>> print(idade['Ana']) # esta linha vai dar erro!
>>> idade.get('Ana') # esta linha não dá erro (não acontece nada)
>>> print(idade.get('Ana'))
None
>>> print(idade.get('Maria'))
18
```

Dicionários

- Para saber se uma chave existe no dicionário, pode-se usar o comando *in*.

```
>>> idade = dict(João=23, Maria=18, Leopoldina=77)
>>> 'Teresa' in idade
False
>>> 'Leopoldina' in idade
True
```

Métodos *items, keys, values*

- ***items()*** retorna uma lista com todos os pares chave/conteúdo.
- ***keys()*** retorna uma lista com todas as chaves.
- ***values()*** retorna uma lista com todos os conteúdos.

```
>>> idade = dict(João=23, Maria=18, Leopoldina=77)
>>> idade.items()
dict_items([('João', 23), ('Maria', 18), ('Leopoldina', 77)])
>>> idade.keys()
dict_keys(['João', 'Maria', 'Leopoldina'])
>>> idade.values()
dict_values([23, 18, 77])
```

Métodos *pop* e *popitem*

- ***pop(chave)***: remove o par chave/conteúdo associado à ***chave*** e retorna o conteúdo.
- ***popitem()*** retorna o último um par chave/conteúdo do dicionário.

```
>>> idade = dict(João=23, Maria=18, Leopoldina=77)
>>> idade.popitem()
('Leopoldina', 77)
>>> idade.pop('Maria')
18
>>> print(idade)
{'João': 23}
```

Arquivos

- Até agora, quando terminamos de executar um programa todos os dados são perdidos!
- Isso acontece pois as variáveis são armazenadas na memória principal (RAM), que é volátil.
- Através do uso de arquivos podemos guardar os dados em memória secundária.
- A memória secundária é persistente!
- Desse modo, quando terminamos de executar o programa os dados são mantidos nos arquivos e podemos recuperá-los em uma nova execução.

Abrir um Arquivo (função *open*)

open(diretorio_nome_arquivo, modo)

- O modo pode ser:
  - 'r': para abrir o arquivo somente para leitura (*read*).
  - 'w': para abrir o arquivo somente escrita (*write*). Caso o arquivo já exista, ele vai ser destruído: um novo arquivo vazio vai ser criado!
  - 'a': o arquivo será aberto somente para escrita, para adicionar dados ao final do arquivo (*append*).
- Podemos também acrescentar o modo 'b' aos três modos anteriores para o arquivo ser tratado como binário.
- Neste curso trabalharemos apenas com arquivos texto.

Abrir um Arquivo (função *open*)

Exemplos:

```
>>> f = open('exemplo.txt', 'w') # vai criar o arquivo no diretório padrão
>>> arq = open('c:/programas/teste.txt', 'w')
```

Fechar um Arquivo (método *close*)

***arquivo*.close()**

- O método ***close()*** é utilizado para fechar um arquivo e liberar recursos.
- Qualquer tentativa de acessar o arquivo ***arquivo*** novamente resultará em erro.
- Só quando se fecha um arquivo você tem certeza de que ele está seguro, i.e., todas as alterações sobre seu conteúdo estarão salvas.
- Caso o programa falhe com o arquivo aberto, seu conteúdo pode ser perdido!

Fechar um Arquivo (método *close*)

Exemplos:

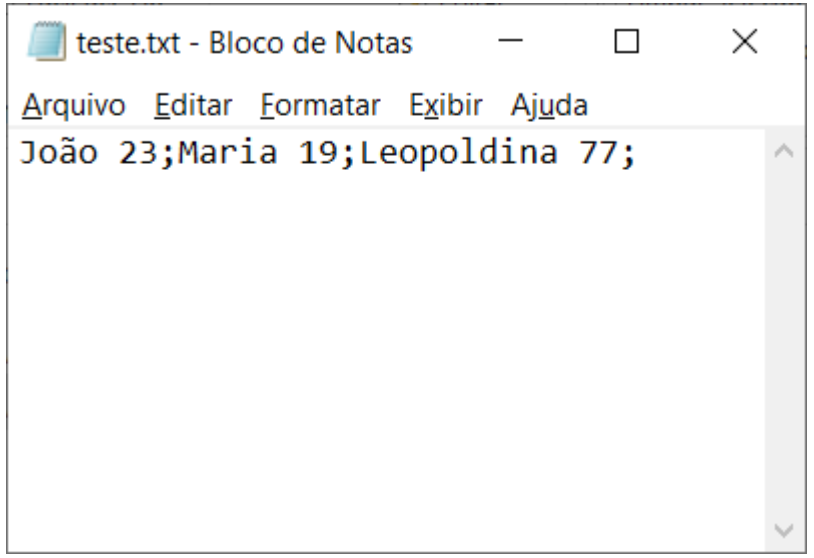
```
>>> f = open('exemplo.txt', 'w')
>>> f.close()
>>> arq = open('c:/programas/teste.txt', 'w')
>>> arq.close()
```

Escrever num Arquivo (método *write*)

***arquivo*.write(*texto*)**

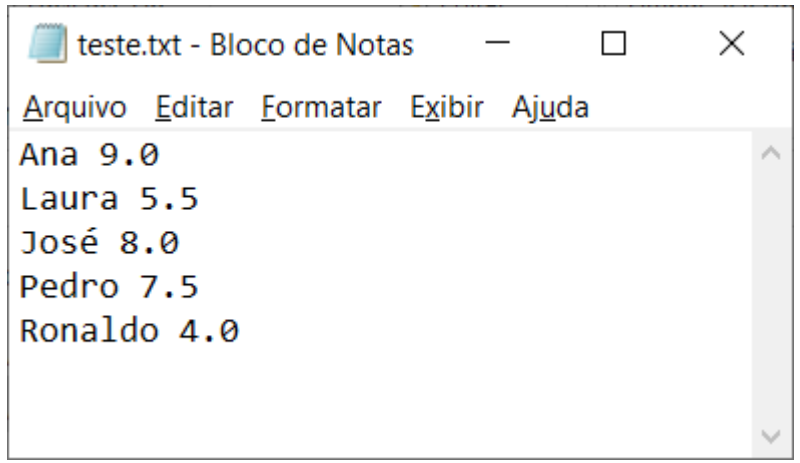
- Escreve o conteúdo de ***texto*** (*string*) no arquivo ***arquivo***.
- Exemplos:

```
>>> arq = open('c:/programas/teste.txt', 'w')
>>> arq.write('João 23;')
8
>>> arq.write('Maria 19;')
9
>>> arq.write('Leopoldina 77;')
14
```



Arquivos

- Fazer um programa para ler cinco nomes e respectivas notas e escrevê-los num arquivo.




```
f = open('c:/programas/teste.txt',"w")
for i in range(0,5):
    nome = input('Nome: ')
    nota = input('Nota: ')
    f.write(nome + ' ' + str(nota)+"\n") # \n vai passar para a prox. linha
f.close()
```

Ler de um Arquivo (métodos *read* e *readline*)

***texto* = *arquivo*.read()**

- Lê todo o conteúdo do arquivo ***arquivo*** e o escreve na variável ***texto*** (*string*).

***texto* = *arquivo*.read(*tamanho*)**

- Lê um certo número (***tamanho***) de caracteres do arquivo escreve na variável ***texto*** (*string*).

***texto* = *arquivo*.readline()**

- Lê uma única linha do arquivo a escreve na variável ***texto*** (*string*).

Ler de um Arquivo (métodos *read* e *readline*)

- Para o método ***read(tamanho)***: quando se lê um número de caracteres do arquivo, a próxima leitura acontecerá a partir da próxima posição do arquivo.
- Para o método ***readline()***: quando se lê uma linha do arquivo, a próxima leitura será a próxima linha.

Método *seek*

- O método ***seek(...)*** pode ser usado para se posicionar dentro do arquivo (para uma nova leitura).
- ***seek(0)*** volta para o início do arquivo.
- ***seek(pos)*** vai para a posição ***pos***.
- Exemplos:

```
>>> nome_do_arquivo.seek(0)
>>> f.seek(10)
```

Método *seek*

- O método ***seek(...)*** pode ser usado para se posicionar dentro do arquivo (para uma nova leitura).
- ***seek(0)*** volta para o início do arquivo.
- ***seek(pos)*** vai para a posição ***pos***.
- ***seek(pos, ref)*** vai para a posição ***pos***, a partir do indicado pela referência ***ref***:
    - ***0***: início do arquivo;
    - ***1***: posição corrente;
    - ***2***: final do arquivo.

Exemplo

Fazer um programa para ler um arquivo e imprimir o conteúdo deste arquivo na tela.

```
f = open('c:/programas/teste.txt',"r")
cadeia = f.read()
print cadeia
f.close()
```

Exemplo

Fazer um programa para ler um arquivo e imprimir o nome do aluno que tirou a maior nota e a média da turma.

```
soma = 0
cont = 0
maiorNota = -1
f = open('c:/programas/teste.txt',"r")
fim = False
while not fim:
    linha = f.readline()
    if len(linha) == 0:
        fim = True
...
```

```
...
    else:
        [nome,nota] = linha.split()
        if float(nota) > maiorNota:
            maiorNota = float(nota)
            maiorNome = nome
        soma += float(nota)
        cont += 1
print('Média:',soma/cont)
print('Maior nota:', maiorNome, maiorNota)
f.close()
```

Exemplo

Fazer um programa para ler um arquivo e imprimir o nome do aluno que tirou a maior nota e a média da turma (outra forma de fazer).

```
soma = 0
cont = 0
maiorNota = -1
f = open('c:/programas/teste.txt',"r")
for linha in f:
    [nome,nota] = linha.split()
    if float(nota) > maiorNota:
        maiorNota = float(nota)
        maiorNome = nome
    soma += float(nota)
    cont += 1
...
```

```
...
print('Média:',soma/cont)
print('Maior nota:', maiorNome, maiorNota)
f.close()
```

Exercício

Fazer um programa para ler um arquivo e criar outro arquivo contendo apenas os nomes dos alunos que tiveram nota acima da média da turma.

Introdução ao Processamento de Dados
Turma 3 (2020.1)

Dicionários e Arquivos

Gilson. A. O. P. Costa (IME/UERJ)

gilson.costa@ime.uerj.br