



# Algoritmos e Lógica de Programação

## *Aula 11*

*Outras Estruturas de Dados – tuplas, dicionários e conjuntos*

Prof. Dr. Dilermando Piva Jr

1º Semestre - DSM



# Tuplas

# Tuplas...

- São coleções (caracteres, strings, números... ) ordenados, separados por vírgula e que estão dentro de parenteses (limitados por parenteses).
- São coleções IMUTÁVEIS

- *Exemplo de Tuplas Homogêneas:*

(1,2,3,4,...)

('a', 'b', 'c', ...)

('texto1', 'texto2', ...)

- *Exemplo de Tuplas Heterogêneas:*

(1, 'a', 'texto1', (1, 2, 3), 345, ...)

# Tuplas...

- Atribuição e tipo
- Operações básicas
- A função tuple()
- Indexação e Fatiamento de Tuplas

# Tuplas – Atribuição e tipo

```
///// ATRIBUIÇÃO E TIPO /////
```

```
# Criando uma Tupla
```

```
# nome_tupla = (item1, item2, ... , itemn)
```

```
montadoras = ("Ferrari", "Fiat", "Mercedes", "Renault")
```

```
montadoras
```

```
("Ferrari", "Fiat", "Mercedes", "Renault")
```

```
type(montadoras)
```

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

# Tuplas – Atribuição e tipo

# Pode-se criar uma tupla vazia.

```
tupla_vazia = ()
```

# As tuplas podem ser heterogêneas

```
tupla_mista = (2022, 34.56, montadoras, "pyPRO")
```

```
tupla_mista
```

```
(2022, 34.56, ('Ferrari', 'Fiat', 'Mercedes', 'Renault'), 'pyPRO')
```

# Tuplas...

- Atribuição e tipo
- Operações básicas
- A função tuple()
- Indexação e Fatiamento de Tuplas

# Tuplas – Operações Básicas

```
//// OPERAÇÕES BÁSICAS  /////
```

```
tupla1 = (2, 0, 1, 4)
```

```
tupla2 = (2, 0, 1, 9)
```

```
tupla1 + tupla2
```

```
tupla1*3
```

```
tupla1 == tupla2
```

```
2 in tupla1
```

```
True
```

```
8 in tupla1
```

```
False
```

```
tupla2 > tupla1
```

```
True
```



# Tuplas...

- Atribuição e tipo
- Operações básicas
- A função `tuple()`
- Indexação e Fatiamento de Tuplas

# Tuplas – a função tuple()

```
///// Função tuple()
navegadores = "vikings"
texto_para_tupla = tuple(navegadores)
texto_para_tupla
('v','i','k','i','n','g','s')
```

```
lista = [23, 34, "Nome", "A", 3.45]
lista_para_tupla = tuple(lista)
(23, 34, 'Nome', 'A', 3.45)
```

```
letras = ('a', 'b', 'c')
numeros = (1, 2, 3)
tuplas_aninhadas = (letras, numeros)
tuplas_aninhadas
(('a', 'b', 'c'), (1, 2, 3))
```

# Tuplas...

- Atribuição e tipo
- Operações básicas
- A função tuple()
- Indexação e Fatiamento de Tuplas

# Tuplas – Indexação e fatiamento

#Fatiamento --> mesmo que as listas

```
cores = ('r', 'g', 'b', 'y', 'o', 'm')
```

```
cores
```

```
('r', 'g', 'b', 'y', 'o', 'm')
```

```
cores[1:4]
```

```
('g', 'b', 'y')
```

```
cores[:3]
```

```
('r', 'g', 'b')
```

```
cores[::]
```

```
('r', 'g', 'b', 'y', 'o', 'm')
```

```
cores[::2]
```

```
cores[::-1]
```

# Tuplas...

- Funções builtins usados em tuplas

Função	Descrição
<code>len()</code>	Retorna o número de elementos de uma tupla
<code>sum()</code>	Retorna a soma dos números de uma tupla
<code>sorted()</code>	Retorna uma cópia modificada (ordenada) de uma tupla e mantém a tupla original intacta.

# Tuplas – Funções Builtins

```
//////// FUNÇÕES BUILTINS
```

```
datas = (1987, 1985, 1981, 2000, 2018, 2022)
```

```
len(datas)
```

```
sum(datas)
```

```
datas_organizadas = sorted(datas)
```

```
datas_organizadas
```

```
(1981, 1985, 1987, 2000, 2018, 2022)
```

# Tuplas...

- Métodos de uma tupla.

Função	Sintaxe	Descrição
<code>count()</code>	<code>Tupla.count(item)</code>	Retorna a quantidade de vezes que um item ocorre em uma tupla
<code>index()</code>	<code>Tupla.index(item)</code>	Busca um item em uma tupla e retorna a sua posição. Se tiver mais de uma ocorrência do item, retorna a primeira. Se não encontrar o item, retorna Erro para o método.

“Tupla” deve ser substituído pelo nome atual da tupla (nome da variável)

# Tuplas – Métodos

```
///// MÉTODOS PARA TUPLAS
```

```
datas.count(1987)
```

```
1
```

```
datas.index(2000)
```

```
3
```



# Tuplas...

Fazer um programa em python que:

- Receba uma quantidade ilimitada (não se sabe quantas) de idade de pessoas.
- Guarde essas idades em uma lista.
- Para encerrar a digitação das idades, deve-se digitar: -1
- Em seguida, deve-se gerar uma tupla de idades, a partir da lista.
- Deve-se mostrar as seguintes informações, consultando a tupla:
  - Qtd de idades digitadas
  - Média das idades

# Resolução...

```
lista_idades = []
while True:
    idade = int(input("Entre com uma idade: "))
    if idade != -1:
        lista_idades.append(idade)
    else:
        break

tupla_idades = tuple(lista_idades)
total = sum(tupla_idades)
qtd = len(tupla_idades)
media = total / qtd
print(f"Total de idades digitadas: {qtd}")
print(f"A média de idades é: {media}")
```

# Dicionários

# Dicionários...

- São coleções de um conjunto de pares chave:valor, com uma restrição de que a chave seja única dentro da coleção.
- Os dicionários são construídos, utilizando chaves {}
- Utiliza-se “:” para separar a chave do valor
- Sempre o operador a esquerda é a chave e o a direita, o valor.

Os dicionários são indexados pela chave.

São as chaves “case sensitive”

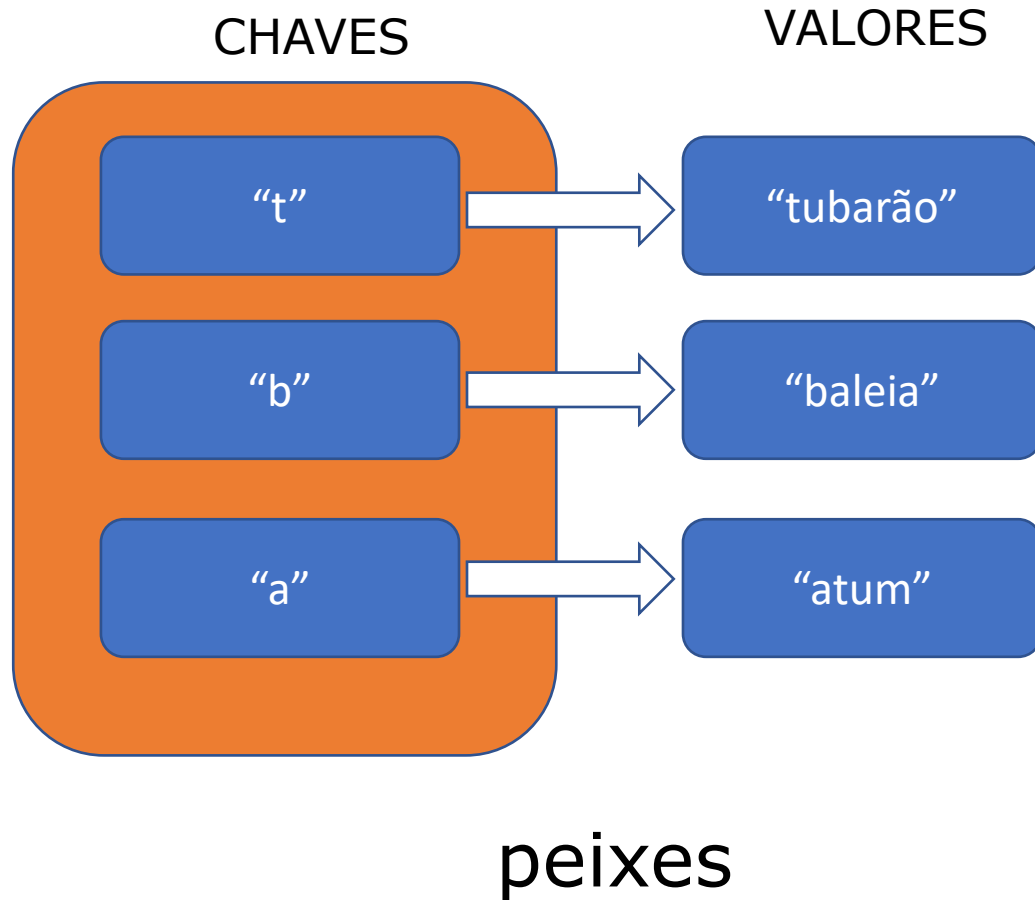
- *Exemplo de Dicionários:*

peixes = {"t": "tubarão", "b": "baleia", "a": "atum"}

f1 = {1: "Mercedes", 2: "Ferrari", 3: "Renault"}

# Dicionários...

peixes = {"t": "tubarão", "b": "baleia", "a": "atum"}



# Dicionários...

- *Exemplo de Dicionários (heterogêneo):*

```
dic_misto = {1:45, 'a':34.56, 'teste':'frase', 3:'pyPRO'}
```

# Dicionários...

- Atribuição e tipo
- Operações básicas
- A função dict()
- Indexação e Alteração

# Dicionários – Atribuição e tipo

```
///// ATRIBUIÇÃO E TIPO /////
```

```
# Criando um Dicionário
```

```
peixes = {"t": "tubarão", "b": "baleia", "a": "atum"}  
f1 = {1: "Mercedes", 2: "Ferrari", 3: "Renault"}
```

```
type(f1)  
<class 'dict'>
```

```
# Pode-se criar um dicionário vazio.
```

```
dic_vazio = {}
```

```
# Os dicionários podem ser heterogêneas
```

```
dic_misto = {1: 45, 'a': 34.56, 'teste': 'frase', 3: 'pyPRO'}
```



# Dicionários...

- Atribuição e tipo
- Operações básicas
- A função dict()
- Indexação e Alteração

# Dicionários – Operações Básicas

```
//// OPERAÇÕES BÁSICAS /////
```

```
peixes = {"t": "tubarão", "b": "baleia", "a": "atum"}
```

```
f1 = {1: "Mercedes", 2: "Ferrari", 3: "Renault"}
```

```
peixes == f1
```

# Dicionários...

- Atribuição e tipo
- Operações básicas
- A função dict()
- Indexação e Alteração

# Dicionários – a função dict()

```
///// Função dict()
```

```
/// dict(**kwarg)
```

```
numeros = dict(um=1, dois=2, tres=3)
```

```
numeros
```

```
{'um':1, 'dois':2, 'tres':3}
```

```
/// dict(iterável[, **kwarg])
```

```
dic1 = dict([('ana', 342), ('pedro', 3453), ('marcio', 8987)])
```

```
dic1
```

```
{'ana':342, 'pedro':3453, 'marcio':8987}
```

# Dicionários...

- Atribuição e tipo
- Operações básicas
- A função dict()
- Indexação e Alteração

# Dicionários – indexação e alteração

```
//// INDEXAÇÃO e ALTERAÇÃO
```

```
dic1
```

```
{'ana':342, 'pedro':3453, 'marcio':8987}
```

```
dic1['ana']
```

```
342
```

```
dic1['piva'] = 3456
```

```
dic1
```

```
{'ana':342, 'pedro':3453, 'marcio':8987, 'piva':3456}
```

```
dic1['jose']
```

```
<<erro>>
```

# Dicionários...

- Funções builtins usados em dicionários

Função	Descrição
<b>len()</b>	Retorna o número de duplas (chave:valor) de um dicionário
<b>all()</b>	Retorna TRUE se todos os valores das chaves do dicionário forem verdadeiras. Caso contrário, FALSE.
<b>any()</b>	Retorna TRUE se pelo menos um valor das chaves do dicionário for verdadeiro. Caso contrário, FALSE.
<b>sorted()</b>	Retorna uma <b>lista</b> modificada (ordenada) de um dicionário baseado em suas chaves. Mantém intacto o dicionário.

# Dicionários – Funções Builtins

///// FUNÇÕES BUILTINS

```
presidentes = {'Deodoro':1889, 'Prudente':1894, 'Washington':1926, 'Vargas':1930,  
'Janio':1961}
```

```
len(presidentes)
```

5

```
dic_logico = {0: True, 1: False, 3:True}
```

```
all(dic_logico)
```

False

```
any(dic_logico)
```

True

```
sorted(presidentes)
```

```
sorted(presidentes, reverse=True)
```

```
sorted(presidentes.values())
```

```
sorted(presidentes.items())
```



# Dicionários...

- Métodos de um dicionário.

Função	Sintaxe	Descrição
<code>clear()</code>	<code>Dic.clear()</code>	Remove todos os pares chave:valor de um dicionário.
<code>fromkeys()</code>	<code>Dic.fromkeys(seq[, valor])</code>	Cria um novo dicionário a partir da sequência dos elementos com um valor informado
<code>get()</code>	<code>Dic.get(chave[, padrão])</code>	Retorna o valor associado à chave especificada no dicionário. Se a chave não estiver presente, ela retornará o valor padrão. Se padrão não for fornecido, o padrão será <i>None</i> , para que esse método nunca gere um <code>KeyError</code> .
<code>items()</code>	<code>Dic.items()</code>	Retorna uma nova visão dos pares de chave e valor do dicionário como tuplas.
<code>keys()</code>	<code>Dic.keys()</code>	Retorna uma nova view que consiste em todas as chaves do dicionário.
<code>pop()</code>	<code>Dic.pop(chave[, padrão])</code>	Remove a chave do dicionário e retorna seu valor. Se a chave não estiver presente, ela retornará o valor padrão. Se o padrão não for fornecido e a chave não estiver no dicionário, isso resultará em <code>KeyError</code> .
<code>popitem()</code>	<code>Dic.popitem()</code>	Remove e retorna um par de tuplas arbitrário (chave, valor) do dicionário. Se o dicionário estiver vazio, chamar <code>popitem()</code> resultará em <code>KeyError</code> .
<code>setdefault()</code>	<code>Dic.setdefault(chave[, padrão])</code>	Retorna um valor para a chave presente no dicionário. Se a chave não estiver presente, insere a chave no dicionário com um valor padrão e retorna o valor padrão. Se a chave estiver presente, o padrão é <i>None</i> , para que esse método nunca gere um <code>KeyError</code> .
<code>update()</code>	<code>Dic.update([outro])</code>	Atualiza o dicionário com os pares chave:valor de outro objeto de dicionário e retorna <i>None</i> .
<code>values()</code>	<code>Dic.values()</code>	Retorna uma nova view que consiste em todos os valores do dicionário.

*"Dic" deve ser substituído pelo nome atual do dicionário (nome da variável)*

# Dicionários - Métodos

```
///// MÉTODOS PARA DICIONÁRIOS
```

```
filmes = {  
    'avatar': 2009,  
    'titanic': 1997,  
    'starwars': 2015,  
    'harry-potter': 2011,  
    'avengers': 2012  
}
```

```
// clear  
filmes.clear()  
filmes
```

# Dicionários - Métodos

```
// fromkeys
```

```
filmes_novachave = filmes.fromkeys(filmes)
```

```
filmes_novachave = filmes.fromkeys(filmes, 'bilionário')
```

```
//get
```

```
print(filmes.get('frozen'))
```

```
print(filmes.get('frozen', 2013))
```

```
print(filmes.get('avatar'))
```

# Dicionários - Métodos

```
//items
```

```
filmes.items()
```

```
// keys
```

```
filmes.keys()
```

```
// values
```

```
filmes.values()
```

```
// update
```

```
filmes.update({"frozen":2013})
```

```
filmes
```

# Dicionários - Métodos

```
// setdefault
```

```
filmes.setdefault("minions")
```

```
filmes
```

```
filmes.setdefault("ironman", 2013)
```

```
filmes
```

```
// pop
```

```
filmes.pop("avatar")
```

```
filmes
```

```
//popitem
```

```
filmes.popitem()
```

# Dicionários – populando um dicionário

//////// PARA POPULAR UM DICIONARIO

```
countries = {}  
countries.update({"Asia": "India"})  
countries.update({"Europa": "Alemanha"})  
countries.update({"Africa": "Egito"})  
countries  
{'Asia': 'India', 'Europa': 'Alemanha', 'Africa': 'Egito'}
```

# Dicionários...

Fazer um programa em python que:

- Receba o cadastro de uma qtd ilimitada (não se sabe quantas) de proprietários e seus respectivos apartamentos (número do apto e nome do proprietário)
- Guarde essas informações em um dicionário, onde a chave de busca é o número do apto.
- Para encerrar a digitação o número do apto será -1
- Em seguida, deve-se mostrar uma listagem em ordem crescente de apto: apto – nome do proprietário
- No final apresente a quantidade total de aptos listados.

# Resolução

```
proprietarios = {}  
while True:  
    apto = int(input("Digite o apto: "))  
    if apto != -1:  
        proprietario = input("Proprietário: ")  
        proprietarios.update({apto: proprietario})  
    else:  
        break  
  
edificio = dict(sorted(proprietarios.items()))  
  
for chave, valor in edificio.items():  
    print(f"{chave} - {valor}")
```



# Conjuntos (sets)

# Conjuntos (sets)

- São coleções não ordenadas (sem índice) que não permitem a duplicação de elementos.
- Os conjuntos são delimitados, utilizando chaves {}
- Os conjuntos suportam as operações matemáticas de união, intersecção, diferença e diferença simétrica.
- São estruturas MUTÁVEIS.

- *Exemplo de Conjuntos (sets)*

peixes = {"tubarão", "baleia", "atum", "sardinha"}

numeros = {1,2,3,6,9,23,67}

mix = {1, 'a', 'nome', 3.56, (2,4,5)}

# Conjuntos...

- Atribuição e tipo
- Operações básicas
- A função set()

# Conjuntos – Atribuição e Tipo

```
cesta = {'maça', 'laranja', 'banana', 'pera', 'laranja', 'maça'}
```

```
cesta
```

```
{'maça', 'laranja', 'pera', 'banana'}
```

```
type(cesta)
```

```
<class 'set'>
```

# Conjuntos...

- Atribuição e tipo
- Operações básicas
- A função set()

# Conjuntos – Atribuição e Tipo

```
'laranja' in cesta
```

```
True
```

```
/// Operações Matemáticas com conjuntos
```

- $a \mid b$  #união entre dois conjuntos
- $a \& b$  #intersecção entre os dois conjuntos
- $a \wedge b$  #diferença entre os dois conjuntos - o que tem de diferente nos dois
- $a - b$  #diferença simétrica entre os dois conjuntos - o que tem no primeiro conjunto que não tem no segundo

# Conjuntos...

- Atribuição e tipo
- Operações básicas
- A função `set()`

# Conjuntos – Atribuição e Tipo

```
// Função set()
```

```
a = set('abracadabra')
```

```
b = set('alacazam')
```

```
a
```

```
b
```



# Conjuntos...

- Funções builtins usados em Conjuntos

Função	Descrição
<code>len()</code>	Retorna o número elementos de um conjunto
<code>sorted()</code>	Retorna uma <b>lista</b> modificada (ordenada) de um conjunto (set). Mantem intacto o conjunto original.

# Conjuntos...

- Métodos de um conjunto(set).

Função	Sintaxe	Descrição
<code>add()</code>	<code>Set.add(item)</code>	Adiciona um item ao conjunto <i>Set</i> .
<code>clear()</code>	<code>Set.clear()</code>	Remove todos os itens do conjunto <i>Set</i> .
<code>difference()</code>	<code>Set.difference(*outros)</code>	Retorna um novo conjunto com itens no conjunto <i>Set</i> que não estão nos <i>outros</i> conjuntos.
<code>discard()</code>	<code>Set.discard(item)</code>	Remove um item do conjunto <i>Set</i> se estiver presente.
<code>intersection()</code>	<code>Set.intersection(*outros)</code>	Retorna um novo conjunto com itens comuns ao conjunto <i>Set</i> e todos os <i>outros</i> conjuntos.
<code>isdisjoint()</code>	<code>Set.isdisjoint(outro)</code>	Retorna True se o conjunto <i>Set</i> não tiver itens em comum com <i>outro</i> conjunto. Conjuntos são disjuntos se e somente se sua interseção for o conjunto vazio.
<code>issubset()</code>	<code>Set.issubset(outro)</code>	Retorna True se todos os itens do conjunto <i>Set</i> estiverem em <i>outro</i> conjunto.

# Conjuntos...

- Métodos de um conjunto(set).

Função	Sintaxe	Descrição
<code>isuperset()</code>	<code>Set.isuperset(outro)</code>	Retorna True se cada elemento em <i>outro</i> conjunto estiver no conjunto <i>Set</i> .
<code>pop()</code>	<code>Set.pop()</code>	Remove e retorna um item arbitrário do conjunto <i>Set</i> . Ele gera <i>KeyError</i> se o conjunto for vazio.
<code>remove()</code>	<code>Set.remove(item)</code>	Remove um item do conjunto <i>Set</i> . Ele gera <i>KeyError</i> se o item não estiver contido no conjunto.
<code>symmetric_difference()</code>	<code>Set.symmetric_difference(outro)</code>	Retorna um novo conjunto com itens no Conjunto <i>Set</i> ou no conjunto <i>outro</i> , mas não em ambos. ((ou exclusivo))
<code>union()</code>	<code>Set.union(*outros)</code>	Retorna um novo conjunto com itens do conjunto <i>Set</i> e todos os <i>outros</i> conjuntos.
<code>update()</code>	<code>Set.update(*outros)</code>	Atualiza o conjunto <i>Set</i> adicionando itens de todos os <i>outros</i> conjuntos.

# Conjuntos - Métodos

```
// Métodos do tipo de dados Conjuntos
```

```
flores_europa = {'rosa', 'lavanda', 'tulipa'}
```

```
flores_america = {'rosa', 'tulipa', 'lirio', 'margarida'}
```

```
// add
```

```
flores_america.add('orquidia')
```

```
// difference
```

```
flores_america.difference(flores_europa)
```

```
// intersection
```

```
flores_america.intersection(flores_europa)
```

# Conjuntos - Métodos

```
// isdisjoint
```

```
flores_america.isdisjoint(flores_europa)
```

```
// issuperset
```

```
flores_america.issuperset(flores_europa)
```

```
// issubset
```

```
flores_america.issubset(flores_europa)
```

```
// symmetric_difference
```

```
flores_america.symmetric_difference(flores_europa)
```

# Conjuntos - Métodos

// union

```
flores_america.union(flores_europa)
```

// discard

```
flores_america.discard('rosa')
```

// pop

```
flores_europa.pop()
```

// clear

```
flores_america.clear()
```

# Dicionários...

Fazer um programa em python que:

- Faça um programa em python que receba uma quantidade variável (não definida) de strings (termine a digitação com string vazia = "")
- Guarde essas informações em uma lista.
- Crie uma outra lista onde cada elemento (palavra) se transformará numa lista de caracteres únicos, ((caracteres não repetidos)). Por exemplo:
- Se uma palavra da lista for igual a: 'barbante' você a modificará para: ['b','a','r','n','t','e'] (tira o que tem repetido)
- Em seguida, mostre a lista modificada.

# Resolução...

```
palavras = []
lista = []

while True:
    palavra = input("Digite uma palavra: ")
    if palavra == '':
        break
    else:
        palavras.append(palavra)

for palavra in palavras:
    lista.append(list(set(palavra)))

for item in lista:
    print(item)
```



# VAMOS PARA A PRÁTICA ?!!!



# EXERCÍCIO 1

**Faça um algoritmo que carregue uma tupla de 10 elementos numéricos inteiros.**

**Após a finalização da entrada, o algoritmo deve escrever a mesma tupla, na ordem inversa de entrada.**

# EXERCÍCIO 2

**Faça um algoritmo que armazenará os 10 primeiros números primos acima de 100 em uma tupla. Ao final, o algoritmo deve mostrar os valores dessa estrutura de dados.**

# EXERCÍCIO 3

**Faça um algoritmo que carregue um dicionário de 10 elementos onde a chave é o sobrenome da pessoa e o valor a sua idade. Após a finalização da entrada, o algoritmo deve escrever o sobrenome da pessoa com maior idade.**

```
# Inicializa um dicionário vazio
idade_por_sobrenome = {}

# Preenche o dicionário com 10 elementos
for _ in range(10):
    sobrenome = input("Digite o sobrenome da pessoa: ")
    idade = int(input("Digite a idade da pessoa: "))
    idade_por_sobrenome[sobrenome] = idade

# Encontra o sobrenome da pessoa mais velha
sobrenome_mais_velho = max(idade_por_sobrenome, key=idade_por_sobrenome.get)

# Exibe o resultado
print(f"\nO sobrenome da pessoa mais velha é: {sobrenome_mais_velho}")
```

# EXERCÍCIO 3a

**Faça um algoritmo que carregue um dicionário contendo nome e idade (pelo menos 5 pares).  
Depois de carregar, mostre todos os nomes que tenham idade maior que a média das idades.**

```
# Inicializa um dicionário vazio  
dados_pessoais = {}
```

```
# Preenche o dicionário com pelo menos 5 pares nome-idade  
for _ in range(5):  
    nome = input("Digite o nome da pessoa: ")  
    idade = int(input("Digite a idade da pessoa: "))  
    dados_pessoais[nome] = idade
```

```
# Calcula a média das idades  
idades = list(dados_pessoais.values())  
media_idades = sum(idades) / len(idades)
```

```
# Mostra os nomes com idade maior que a média  
print("\nNomes com idade maior que a média:")  
for nome, idade in dados_pessoais.items():  
    if idade > media_idades:  
        print(f"{nome}: {idade} anos")
```

# EXERCÍCIO 3

**Faça algoritmo que carregue duas listas de cinco elementos numéricos inteiros cada um. A partir dessas duas listas, crie um conjunto da união entre essas duas listas.**



```
# Carrega duas listas de cinco elementos numéricos inteiros cada
lista1 = [int(input(f"Digite o {i+1}º elemento da primeira lista: ")) for i in range(5)]
lista2 = [int(input(f"Digite o {i+1}º elemento da segunda lista: ")) for i in range(5)]

# Cria um conjunto da união entre as duas listas
conjunto_uniao = set(lista1).union(lista2)

# Exibe o resultado
print("\nConjunto da união entre as duas listas:", conjunto_uniao)
```