

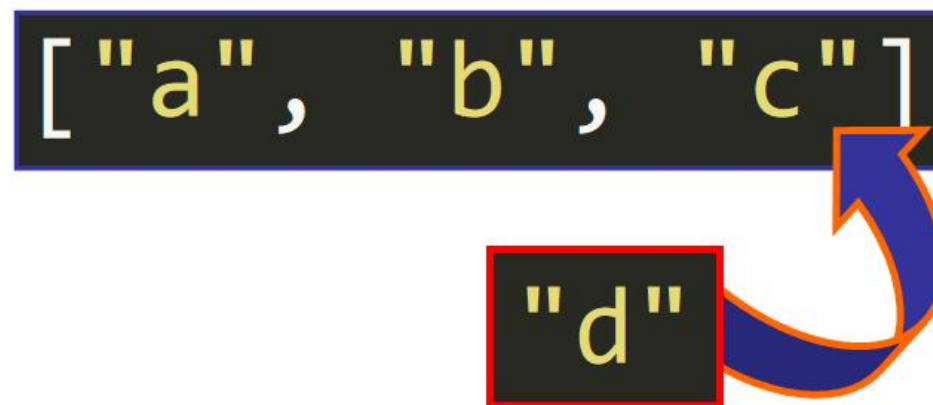
# LÓGICA E LINGUAGEM DE PROGRAMAÇÃO

Curso Técnico Integrado em Informática  
Lucas Sampaio Leite



# Listas

- Em Python, uma lista é uma estrutura de dados que permite armazenar uma coleção de elementos, que podem ser de tipos diferentes (números, strings, booleanos, outras listas etc.).
- As listas são mutáveis, ou seja, seus elementos podem ser alterados após a criação.



# Listas

- Listas são baseadas na ideia de vetor, que é bastante comum na matemática, onde utilizamos variáveis com índices subscritos para representar conjuntos de valores relacionados.
- Exemplo:
  - Matematicamente:  $x_1, x_2, x_3, \dots, x_n$
  - Em programação: usamos um índice numérico para acessar cada elemento. Ex:  $x[0], x[1], x[2], \dots, x[n]$

# Listas

- Em Python, uma lista é uma sequência mutável de elementos, capaz de armazenar n valores de qualquer tipo, incluindo até mesmo outras listas.
- De forma geral, pode-se entender uma lista como um vetor de elementos heterogêneos, ou seja, com valores de diferentes tipos.
- Em termos de manipulação, as listas são muito semelhantes às strings, pois ambas permitem acesso por índice, fatiamento e iteração. A principal diferença é que as strings são imutáveis, enquanto as listas podem ser modificadas após a criação.

```
lista = ["Pode conter qualquer tipo de valor", 0, 5 + 8j,  
         ["Outras listas", "por exemplo"], 5.5]
```

# Listas

- Uma lista é um agrupamento de elementos, ou seja, uma coleção de valores organizados em sequência.
- Como criar uma lista em Python?
  - Utiliza-se colchetes ([ ]) para definir a lista e separamos os elementos por vírgulas.
  - Exemplo: Lista contendo os números ímpares positivos menores que 10:

```
lista = [1, 3, 5, 7, 9]
```

# Listas

- Cada elemento de uma lista pode ser acessado individualmente por meio de acesso indexado.
- Para isso, utiliza-se o nome da lista seguido do índice entre colchetes ([ ]), indicando a posição do elemento desejado.

- Exemplo:

```
lista = [1, 3, 5, 7, 9]  
print(lista[0])
```

# Listas

- Cada elemento de uma lista pode ser acessado individualmente por meio de acesso indexado.
- Para isso, utiliza-se o nome da lista seguido do índice entre colchetes ([ ]), indicando a posição do elemento desejado.

- Exemplo:

```
lista = [1, 3, 5, 7, 9]  
print(lista[0])
```

- A contagem de posições em listas começa em zero, seguindo a seguinte lógica:
- Posição 0 → Primeiro elemento
- Posição 1 → Segundo elemento
- Posição 2 → Terceiro elemento
- ... e assim sucessivamente.

# Listas

- Cada elemento de uma lista pode ser acessado individualmente por meio de acesso indexado.
- Para isso, utiliza-se o nome da lista seguido do índice entre colchetes ([ ]), indicando a posição do elemento desejado.

- Exemplo:

```
lista = [1, 3, 5, 7, 9]  
print(lista[0])
```

- A contagem de posições em listas começa em zero, seguindo a seguinte lógica:
- Posição 0 → Primeiro elemento
- Posição 1 → Segundo elemento
- Posição 2 → Terceiro elemento
- ... e assim sucessivamente.



Cuidado! Em uma lista com “n” elementos, a última posição é a “n-1”.



# Listas

- A função embutida (*built-in*) `len()` em Python é utilizada para obter a quantidade de elementos presentes em uma lista.

```
lista = [1, 3, 5, 7, 9]  
print(lista)  
print(type(lista))  
print(len(lista))
```



```
[1, 3, 5, 7, 9]  
<class 'list'>  
5
```

# Listas

- O que será impresso?

```
lista = [1, 3, 5, 7, 9]
print(lista[0])
print(lista[3])
print(lista[4])
print(lista[len(lista)-1])
print(lista[len(lista)-3])
print(len(lista))
```

# Listas

- O que será impresso?

```
lista = [1, 3, 5, 7, 9]
print(lista[0])
print(lista[3])
print(lista[4])
print(lista[len(lista)-1])
print(lista[len(lista)-3])
print(len(lista))
```



1  
7  
9  
9  
5  
5

# Listas

- O que será impresso?

```
lista = [1, 3, 5, 7, 9]  
print(lista[5])
```

```
lista = [1, 3, 5, 7, 9]  
print(lista[len(lista)])
```

# Listas

- O que será impresso?
  - Índice inexistente.
  - ERRO!!! IndexError: list index out of range.

```
lista = [1, 3, 5, 7, 9]  
print(lista[5])
```



```
lista = [1, 3, 5, 7, 9]  
print(lista[len(lista)])
```



Os índices válidos correspondem às posições de 0 a 4.

# Listas

- É possível adicionar novos elementos a uma lista.
- Uma das formas de fazer isso é utilizando o operador de concatenação (+), que permite juntar a lista original com outra lista de novos elementos.
- Os novos elementos são sempre adicionados ao final da lista.

```
lista = [1, 3, 5, 7, 9]
print("Lista antes da primeira concatenação: ", lista)
lista = lista + [11, 13]
print("Lista após a primeira concatenação: ", lista)
lista += [15, 17, 19]
print("Lista após a segunda concatenação: ", lista)
```



# Listas

- É possível adicionar novos elementos a uma lista.
- Uma das formas de fazer isso é utilizando o operador de concatenação (+), que permite juntar a lista original com outra lista de novos elementos.
- Os novos elementos são sempre adicionados ao final da lista.

```
lista = [1, 3, 5, 7, 9]
print("Lista antes da primeira concatenação: ", lista)
lista = lista + [11, 13]
print("Lista após a primeira concatenação: ", lista)
lista += [15, 17, 19]
print("Lista após a segunda concatenação: ", lista)
```

```
Lista antes da primeira concatenação:  [1, 3, 5, 7, 9]
Lista após a primeira concatenação:  [1, 3, 5, 7, 9, 11, 13]
Lista após a segunda concatenação:  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

# Listas

- Listas são estruturas mutáveis, portanto, é possível alterar os valores de seus elementos.

```
lista = [1, 3, 5, 7, 9]
print("Lista antes das alterações: ", lista)
lista[0] = 0
lista[2] = 4
print("Lista após as alterações: ", lista)
```



# Listas

- Listas são estruturas mutáveis, portanto, é possível alterar os valores de seus elementos.

```
lista = [1, 3, 5, 7, 9]
print("Lista antes das alterações: ", lista)
lista[0] = 0
lista[2] = 4
print("Lista após as alterações: ", lista)
```



```
Lista antes das alterações:  [1, 3, 5, 7, 9]
Lista após as alterações:   [0, 3, 4, 7, 9]
```

# Listas

- Fatiamento de listas é o processo de extrair um subconjunto dos elementos de uma lista.
- A notação utilizada é semelhante ao acesso indexado, usando colchetes [].
- Contudo, em vez de informar uma única posição, especifica-se um intervalo de posições.
- Para definir esse intervalo, utiliza-se a sintaxe:

```
lista = [1, 3, 5, 7, 9]  
print(lista[2:4])
```

O início é o índice da posição inicial (inclusivo)  
O fim é o índice da posição final (exclusivo)



# Listas

- O fatiamento de uma lista retorna uma nova lista contendo os elementos selecionados.
- Isso significa que o resultado do fatiamento pode ser armazenado em uma variável diferente para uso posterior.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista: ", lista)
sublista = lista[0:6]
print("Sublista: ", sublista)
```

# Listas

- O fatiamento de uma lista retorna uma nova lista contendo os elementos selecionados.
- Isso significa que o resultado do fatiamento pode ser armazenado em uma variável diferente para uso posterior.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
print("Lista: ", lista)  
sublista = lista[0:6]  
print("Sublista: ", sublista)
```



```
Lista:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
Sublista:  [1, 2, 3, 4, 5, 6]
```

# Listas

- É possível definir um intervalo entre os elementos do subconjunto obtido por fatiamento.
- Para isso, utiliza-se a seguinte notação: `lista[início:fim:passo]`
- Dessa forma, o fatiamento seleciona elementos desde a posição início até imediatamente antes da posição fim, avançando de passo em passo (ou seja, pulando elementos conforme o valor do passo).

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista: ", lista)
sublista = lista[0:10:2]
print("Sublista: ", sublista)
```

# Listas

- É possível definir um intervalo entre os elementos do subconjunto obtido por fatiamento.
- Para isso, utiliza-se a seguinte notação: `lista[início:fim:passo]`
- Dessa forma, o fatiamento seleciona elementos desde a posição início até imediatamente antes da posição fim, avançando de passo em passo (ou seja, pulando elementos conforme o valor do passo).

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista: ", lista)
sublista = lista[0:10:2]
print("Sublista: ", sublista)
```



```
Lista: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sublista: [1, 3, 5, 7, 9]
```



# Listas

- No fatiamento `lista[início:fim:passo]`, os valores podem ser omitidos.
- Python adota os seguintes valores padrão:
  - início = 0 → início do fatiamento na posição zero (primeiro elemento);
  - fim = n → final do fatiamento na posição imediatamente após o último elemento, onde n é o número total de elementos da lista;
  - passo = 1 → passo igual a 1, ou seja, sem pular elementos.

# Listas

- O que será impresso?



```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista:", lista)
print(lista[::])
print(lista[5::])
print(lista[:5:])
print(lista[::2])
```



# Listas

- O que será impresso?

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
print("Lista:", lista)  
print(lista[::])  
print(lista[5::])  
print(lista[:5:])  
print(lista[::2])
```



```
Lista: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
[6, 7, 8, 9, 10]  
[1, 2, 3, 4, 5]  
[1, 3, 5, 7, 9]
```

# Listas

- É possível utilizar o fatiamento para alterar os elementos de uma lista.
- A mesma regra usada para definir o subconjunto da lista no fatiamento é aplicada.
- No entanto, nesse caso, a lista e o intervalo desejado devem ser especificados à esquerda do operador de atribuição (=), indicando os elementos que serão substituídos.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista antes da modificação:", lista)
lista[3:8] = []
print("Lista após a modificação:", lista)
```

# Listas

- É possível utilizar o fatiamento para alterar os elementos de uma lista.
- A mesma regra usada para definir o subconjunto da lista no fatiamento é aplicada.
- No entanto, nesse caso, a lista e o intervalo desejado devem ser especificados à esquerda do operador de atribuição (=), indicando os elementos que serão substituídos.

O código atribui um subconjunto vazio à fatia que vai da posição 3 até imediatamente antes da posição 7.



```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista antes da modificação:", lista)
lista[3:7] = []
print("Lista após a modificação:", lista)
```

# Listas

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
print("Lista antes da modificação:", lista)  
lista[3:8] = [5, 8]  
print("Lista após a modificação:", lista)
```



```
Lista antes da modificação: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
Lista após a modificação: [1, 2, 3, 5, 8, 9, 10]
```

# Listas

- O fatiamento também pode ser utilizado para remover um elemento ou um subconjunto de elementos da lista.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista antes da remoção:", lista)
lista[2:3] = []
lista[2:5] = []
print("Lista após as remoções:", lista)
```

O que será impresso?



# Listas

- O fatiamento também pode ser utilizado para remover um elemento ou um subconjunto de elementos da lista.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista antes da remoção:", lista)
lista[2:3] = []
lista[2:5] = []
print("Lista após as remoções:", lista)
```



```
Lista antes da remoção: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Lista após as remoções: [1, 2, 7, 8, 9, 10]
```

# Listas

- A remoção de elementos também pode ser realizada utilizando o comando `del`.
- As posições dos elementos a serem removidos devem ser especificadas entre colchetes, utilizando o índice ou um intervalo de fatiamento.

O que será  
impresso?

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista antes das remoções:", lista)
del lista[0]
print("Lista após a primeira remoção:", lista)
del lista[8]
print("Lista após a segunda remoção:", lista)
del lista[2:7]
print("Lista após a terceira remoção:", lista)
```

# Listas

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Lista antes das remoções:", lista)
del lista[0]
print("Lista após a primeira remoção:", lista)
del lista[8]
print("Lista após a segunda remoção:", lista)
del lista[2:7]
print("Lista após a terceira remoção:", lista)
```



```
Lista antes das remoções: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Lista após a primeira remoção: [2, 3, 4, 5, 6, 7, 8, 9, 10]
Lista após a segunda remoção: [2, 3, 4, 5, 6, 7, 8, 9]
Lista após a terceira remoção: [2, 3, 9]
```



# Listas

- Existem outras formas de remover elementos de uma lista em Python:
  - O método `pop()`, que remove um elemento com base em seu índice e retorna o valor removido.
  - O método `remove()` exclui o primeiro elemento da lista que corresponda ao valor especificado.

O que será  
impresso?

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = lista.pop(5)
print(x)
lista.remove(4)
print("Lista após as remoções:", lista)
```

# Listas

- Existem outras formas de remover elementos de uma lista em Python:
  - O método `pop()`, que remove um elemento com base em seu índice e retorna o valor removido.
  - O método `remove()` exclui o primeiro elemento da lista que corresponda ao valor especificado.

O que será  
impresso?

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = lista.pop(5)
print(x)
lista.remove(4)
print("Lista após as remoções:", lista)
```

6

Lista após as remoções: [1, 2, 3, 5, 7, 8, 9, 10]



# Listas

- O método `append()` é utilizado para adicionar um novo elemento ao final da lista.

```
lista = [1, 3, 5, 7, 9]
print("Lista antes das inserções:", lista)
lista.append("IF")
lista.append("Baiano")
lista.append(["a", "b"])
print("Lista após as inserções:", lista)
```



```
Lista antes das inserções: [1, 3, 5, 7, 9]
Lista após as inserções: [1, 3, 5, 7, 9, 'IF', 'Baiano', ['a', 'b']]
```

# Listas

- O método `insert(índice, valor)` insere um elemento em uma posição específica da lista.

O que será  
impresso?

```
lista = [1, 3, 5, 7, 9]
print("Lista antes das inserções:", lista)
lista.insert(0, "IF")
print("Lista após primeira inserção:", lista)
lista.insert(-1, "Baiano")
print("Lista após segunda inserção:", lista)
lista.insert(len(lista), "Bonfim")
print("Lista após terceira inserção:", lista)
```



# Listas

- O método `insert(índice, valor)` insere um elemento em uma posição específica da lista.

O que será  
impresso?

```
lista = [1, 3, 5, 7, 9]
print("Lista antes das inserções:", lista)
lista.insert(0, "IF")
print("Lista após primeira inserção:", lista)
lista.insert(-1, "Baiano")
print("Lista após segunda inserção:", lista)
lista.insert(len(lista), "Bonfim")
print("Lista após terceira inserção:", lista)
```

```
Lista antes das inserções: [1, 3, 5, 7, 9]
Lista após primeira inserção: ['IF', 1, 3, 5, 7, 9]
Lista após segunda inserção: ['IF', 1, 3, 5, 7, 'Baiano', 9]
Lista após terceira inserção: ['IF', 1, 3, 5, 7, 'Baiano', 9, 'Bonfim']
```



# Listas

- Um elemento de uma lista também pode ser outra lista.
- O resultado é conhecido como lista aninhada (ou *nested list*).

```
lista = ["Lista heterogênea", 1, 2, 3, [0.5, 1.75, 9.55], True]  
print("Lista alinhada:", lista)  
print("Tamanho da lista:", len(lista))
```

# Listas

- Um elemento de uma lista também pode ser outra lista.
- O resultado é conhecido como lista aninhada (ou *nested list*).

```
lista = ["Lista heterogênea", 1, 2, 3, [0.5, 1.75, 9.55], True]
print("Lista alinhada:", lista)
print("Tamanho da lista:", len(lista))
```



```
Lista alinhada: ['Lista heterogênea', 1, 2, 3, [0.5, 1.75, 9.55], True]
Tamanho da lista: 6
```

# Listas

```
lista = ["Lista heterogênea", 1, 2, 3, [0.5, 1.75, 9.55], True]
print("Lista alinhada:", lista)
print("Tamanho da lista:", len(lista))
print("Elemento da posição 4:", lista[4])
print("Primeiro elemento da posição 4:", lista[4][0])
print("Último elemento da posição 4:", lista[4][-1])
print("Último elemento da posição 4:", lista[4][len(lista[4])-1])
print("Tamanho do elemento da posição 4:", len(lista[4]))
```



# Listas

```
lista = ["Lista heterogênea", 1, 2, 3, [0.5, 1.75, 9.55], True]
print("Lista alinhada:", lista)
print("Tamanho da lista:", len(lista))
print("Elemento da posição 4:", lista[4])
print("Primeiro elemento da posição 4:", lista[4][0])
print("Último elemento da posição 4:", lista[4][-1])
print("Último elemento da posição 4:", lista[4][len(lista[4])-1])
print("Tamanho do elemento da posição 4:", len(lista[4]))
```

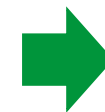


```
Lista alinhada: ['Lista heterogênea', 1, 2, 3, [0.5, 1.75, 9.55], True]
Tamanho da lista: 6
Elemento da posição 4: [0.5, 1.75, 9.55]
Primeiro elemento da posição 4: 0.5
Último elemento da posição 4: 9.55
Último elemento da posição 4: 9.55
Tamanho do elemento da posição 4: 3
```

# Listas

- Algumas funções para operar sobre listas com valores numéricos:
  - `sum(lista)`: retorna a soma de todos os elementos da lista;
  - `min(lista)`: retorna o valor mínimo armazenado;
  - `max(lista)`: retorna o valor máximo armazenado;
  - `lista.count(valor)`: retorna o número de ocorrências de valor em lista:

```
lista = [10, 1.5, 2, 13, 4, 85, 10]  
print(sum(lista))  
print(min(lista))  
print(max(lista))  
print(lista.count(10))
```



```
125.5  
1.5  
85  
2
```

# Listas

- O método `sort()` é utilizado para ordenar os elementos de uma lista, de acordo com a ordem natural dos elementos (numérica ou alfabética).
- A ordenação é feita "in place", ou seja, a própria lista original é modificada.

```
lista_numeros = [10, 1.5, 2, 13, 4, 85, 10]
lista_numeros.sort()
print(lista_numeros)

lista_nomes = ["Ziraldo", "Lucas", "Marcos", "Ana"]
lista_nomes.sort(reverse = True)
print(lista_nomes)
```



```
[1.5, 2, 4, 10, 10, 13, 85]
['Ziraldo', 'Marcos', 'Lucas', 'Ana']
```





# Listas

- Ordenando uma lista:

```
lista_numeros = [10, 1.5, 2, 13, 4, 85, 10]  
lista_numeros.sort()  
print(lista_numeros)
```

```
lista_nomes = ["Ziraldo", "Lucas", "Marcos", "Ana"]  
lista_nomes.sort(reverse = True)  
print(lista_nomes)
```



```
[1.5, 2, 4, 10, 10, 13, 85]  
['Ziraldo', 'Marcos', 'Lucas', 'Ana']
```

# Listas

- A função embutida `sorted()` é utilizada para ordenar elementos de qualquer estrutura iterável.
- Não altera o conteúdo original da estrutura de dados.

```
lista_numeros = [10, 1.5, 2, 13, 4, 85, 10]
print(sorted(lista_numeros))
print(lista_numeros)
```

```
lista_nomes = ["Ziraldo", "Lucas", "Marcos", "Ana"]
print(sorted(lista_nomes, reverse = True))
print(lista_nomes)
```

# Listas

```
lista_numeros = [10, 1.5, 2, 13, 4, 85, 10]
print(sorted(lista_numeros))
print(lista_numeros)

lista_nomes = ["Ziraldo", "Lucas", "Marcos", "Ana"]
print(sorted(lista_nomes, reverse = True))
print(lista_nomes)
```



```
[1.5, 2, 4, 10, 10, 13, 85]
[10, 1.5, 2, 13, 4, 85, 10]
['Ziraldo', 'Marcos', 'Lucas', 'Ana']
['Ziraldo', 'Lucas', 'Marcos', 'Ana']
```



# Listas

- Percorrendo uma lista usando for direto, iterando por valor:

```
frutas = ["maçã", "banana", "cereja", "damasco"]  
  
for fruta in frutas:  
    print(fruta)
```



```
maçã  
banana  
cereja  
damasco
```

# Listas

- Percorrendo uma lista usando for com range(), iterando pelo índice:

```
frutas = ["maçã", "banana", "cereja", "damasco"]  
  
for i in range(len(frutas)):  
    print(f"frutas[{i}] = {frutas[i]}")
```



```
frutas[0] = maçã  
frutas[1] = banana  
frutas[2] = cereja  
frutas[3] = damasco
```

# Listas

- Percorrendo uma lista usando while:

```
frutas = ["maçã", "banana", "cereja", "damasco"]  
  
i = 0  
while i < len(frutas):  
    print(frutas[i])  
    i += 1
```



```
maçã  
banana  
cereja  
damasco
```

# Exercícios

1. Faça um Programa que leia 5 números inteiros, armazene-os em uma lista e imprima essa lista na tela.
2. Faça um programa que leia 10 números inteiros. Cada numero par deve ser armazenado em uma lista de pares e cada impar tem que ser armazenado em uma lista de impares. Ao término do programa imprima as duas listas.
3. Faça um programa que armazene as idades e as alturas de 4 alunos. Seu programa deve exibir quantos alunos com mais de 13 anos possuem uma altura inferior à altura média dentre todos os alunos.

# Exercícios

4. Modifique o programa da questão 3 para que o usuário indique a quantidade de alunos que será utilizada no programa. Assim antes de começar a leitura de idades e alturas, o programa deve solicitar ao usuário o quantitativo de alunos.
5. Modifique o programa da questão 3 para que o programa funcione para qualquer quantidade de alunos. Assim, durante a leitura das idades e alturas o usuário poderá inserir um valor negativo para indicar que deseja interromper a leitura dos dados.

## Exercícios

6. Faça um programa que leia uma data de nascimento no formato dd/mm/aaaa e imprima a data com o mês escrito por extenso.

Exemplo: Data = 20/02/1995

Resultado gerado pelo programa: Você nasceu em 20 de fevereiro de 1995

7. Faça um programa que receba a temperatura média de cada mês do ano e armazene-as em uma lista. Após isto, calcule a média anual das temperaturas e mostre todas as temperaturas acima da média anual, e em que mês elas ocorreram (mostrar o mês por extenso: 1 - Janeiro, 2 - Fevereiro, . . . ).



## Exercícios

8. Em uma competição de salto em distância cada atleta tem direito a cinco saltos. O resultado do atleta será determinado pela média dos cinco valores restantes. Você deve fazer um programa que receba o nome e as cinco distâncias alcançadas pelo atleta em seus saltos e depois informe o nome, os saltos e a média dos saltos. O programa deve ser encerrado quando não for informado o nome do atleta. A saída do programa deve ser conforme ao lado.

Atleta: Rodrigo Curvêllo

Primeiro Salto: 6.5 m

Segundo Salto: 6.1 m

Terceiro Salto: 6.2 m

Quarto Salto: 5.4 m

Quinto Salto: 5.3 m

Resultado final:

Atleta: Rodrigo Curvêllo

Saltos: 6.5 - 6.1 - 6.2 - 5.4 - 5.3

Média dos saltos: 5.9 m

# Dúvidas



# LÓGICA E LINGUAGEM DE PROGRAMAÇÃO

Curso Técnico Integrado em Informática  
Lucas Sampaio Leite

