Listas

CIÊNCIA DA COMPUTAÇÃO

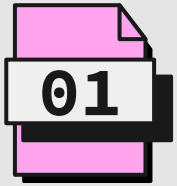
ATITUS EDUCAÇÃO

- Aula 08 -
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto

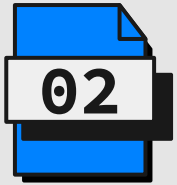
ATITUS EDUCAÇÃO

Na aula de hoje...



Listas

Definição
Nomenclatura
Declaração

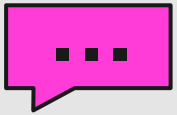


Operações com Listas

Modificação
Inserção de Elementos
Remoção de Elementos
Iterando sobre listas

Operações Avançadas com Listas

Remoção por conteúdo
Tamanho da Lista
Contar ocorrências de valores
Ordenar Listas
Limpar uma lista



Problema 0: Notas

Enunciado de um problema:

Ler as notas finais de 30 alunos. Calcular e informar a média da turma

Implementação do Programa

```
ALUNOS = 30

soma = 0
for aluno in range(ALUNOS):
    nota = float(input("Informe a nota: "))
    soma += nota

media = soma / ALUNOS
print(f"MÉDIA DA TURMA: {media:.2f}")
```

Problema 0: Notas Parte 2

Enunciado de um problema:

Ler as notas finais de 30 alunos. Calcular e informar a média da turma. Informar as notas que são superiores à média calculada.

```python
ALUNOS = 30

# Obtém a nota de cada aluno
n01 = float(input("Informe a nota: "))
n02 = float(input("Informe a nota: "))
n03 = float(input("Informe a nota: "))
n04 = float(input("Informe a nota: "))
n05 = float(input("Informe a nota: "))
n06 = float(input("Informe a nota: "))
n07 = float(input("Informe a nota: "))
n08 = float(input("Informe a nota: "))
n09 = float(input("Informe a nota: "))
n10 = float(input("Informe a nota: "))
n11 = float(input("Informe a nota: "))
n12 = float(input("Informe a nota: "))
n13 = float(input("Informe a nota: "))
n14 = float(input("Informe a nota: "))
n15 = float(input("Informe a nota: "))
n16 = float(input("Informe a nota: "))
n17 = float(input("Informe a nota: "))
n18 = float(input("Informe a nota: "))
n19 = float(input("Informe a nota: "))
n20 = float(input("Informe a nota: "))
n21 = float(input("Informe a nota: "))
n22 = float(input("Informe a nota: "))
n23 = float(input("Informe a nota: "))
n24 = float(input("Informe a nota: "))
n25 = float(input("Informe a nota: "))
n26 = float(input("Informe a nota: "))
n27 = float(input("Informe a nota: "))
n28 = float(input("Informe a nota: "))
n29 = float(input("Informe a nota: "))
n30 = float(input("Informe a nota: "))

# soma todas as notas e calcula a média
soma = n01 + n02 + n03 + n04 + n05 + n06 + n07 + n08 + n09 + n10 + n11 + n12 + n13 +
n14 + n15 + n16 + n17 + n18 + n19 + n20 + n21 + n22 + n23 + n24 + n25 + n26 + n27 +
n28 + n29 + n30
media = soma / ALUNOS

# Escreve a média
print(f"Media da turma: {media:.2f}")

# Escreve as notas acima da média
if n01 > media: print(n01)
if n02 > media: print(n02)
if n03 > media: print(n03)
if n04 > media: print(n04)
if n05 > media: print(n05)
if n06 > media: print(n06)
if n07 > media: print(n07)
if n08 > media: print(n08)
if n09 > media: print(n09)
if n10 > media: print(n10)
if n11 > media: print(n11)
if n12 > media: print(n12)
if n13 > media: print(n13)
if n14 > media: print(n14)
if n15 > media: print(n15)
if n16 > media: print(n16)
if n17 > media: print(n17)
if n18 > media: print(n18)
if n19 > media: print(n19)
if n20 > media: print(n20)
if n21 > media: print(n21)
if n22 > media: print(n22)
if n23 > media: print(n23)
if n24 > media: print(n24)
if n25 > media: print(n25)
if n26 > media: print(n26)
if n27 > media: print(n27)
if n28 > media: print(n28)
if n29 > media: print(n29)
if n30 > media: print(n30)
```

E se fossem 1000 alunos?
Como generalizar?

Na aula de hoje...

Como armazenar vários valores sob uma mesma variável?

Listas! (list)

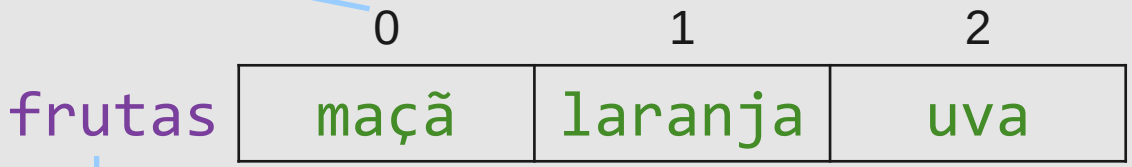
```
>>> frutas = ['maçã', 'laranja', 'uva']
>>> type(frutas)
<class 'list'>
```

Intuitivamente, podemos pensar em listas como uma **coleção de dados** que podem ser de qualquer um dos tipos básicos ou até mesmo outras listas.

Listas - Nomenclatura

O primeiro índice é **sempre** zero

ÍNDICES: Identificação de cada posição da lista



NOME: Comum para todos os elementos

TAMANHO: Quantidade máxima de valores que podem ser armazenados

Definição

Uma **lista** é uma **coleção ordenada** de dados que é referida por um **único nome** de variável.

(Outras linguagens de programação se referem ao mesmo conceito como arranjo ou vetor).

--Irv Kalb, Learn to Program with Python.

* Essa relação de **ordem** entre os elementos tem a ver com a sua posição na lista e não com seu valor
  - Se existe um elemento na primeira posição, outro na segunda e assim por diante, essa relação será sempre mantida por essa estrutura de dados
  - Cada elemento terá um **índice** pelo qual se pode acessar o seu valor
* Os dados armazenados em uma lista, em geral, estão semanticamente relacionados

Exemplos

* Coleções de dados semanticamente relacionados que poderiam ser armazenados em listas
  - → Nomes dos times de um campeonato de futebol
    - Lista de palavras/texto (`str`)
  - → Notas dos alunos de uma turma
    - Lista de números reais (`float`)
  - → Idade de pessoas participando de uma pesquisa
    - Lista de inteiros (`int`)
  - → Resultados de uma bateria de testes (sucesso/falha)
    - Lista de booleanos (`bool`)
  - → Nomes dos produtos e preços de um pedido
    - Lista mista de texto e números reais

É possível misturar dados de vários tipos em uma lista, mas não é muito recomendável.

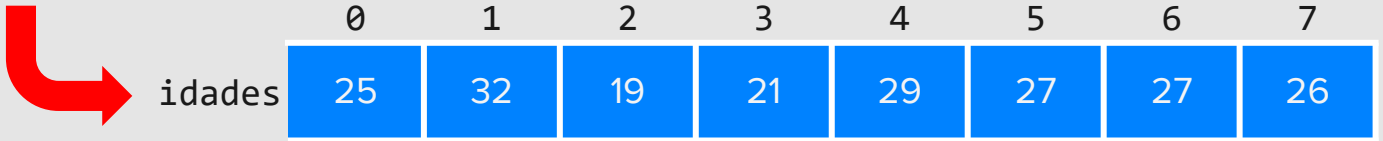
Declarando Listas

Lista vazia:

```
lista = []
```

Idade de pessoas participando de uma pesquisa:

```
idades = [25, 32, 19, 21, 29, 27, 27, 26]
```



Nomes dos times de um campeonato de futebol:

```
times = ["Grêmio", "Inter", "Flamengo", "Palmeiras", ..., "Santos"]
```

Inicializando uma lista

Lista vazia:

```
lista = []
```

Com valores pré-definidos:

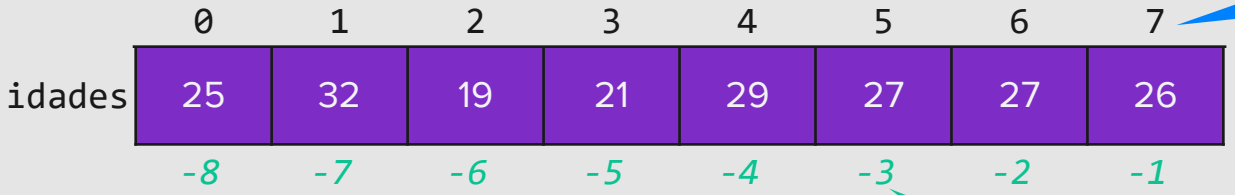
```
idades = [25, 32, 19, 21, 29, 27, 27, 26]
```

Com um valor padrão

```
idades = [0] * 10
```

Lista de 10 elementos inicializada com zeros

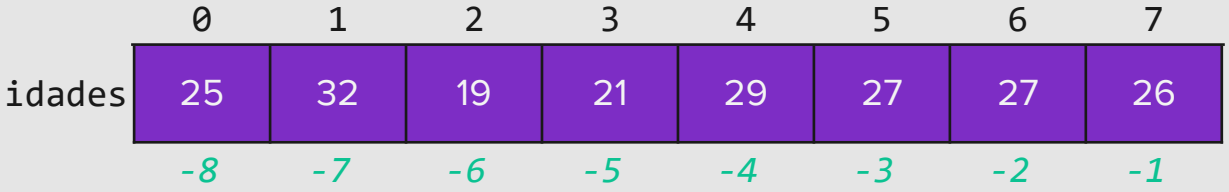
Acessando Elementos da Lista



Para acessar cada elemento utilizamos seu **índice**.

Índices negativos também são válidos. Para acessar o **penúltimo** elemento podemos usar **`idade[-2]`**.

Acessando Elementos da Lista



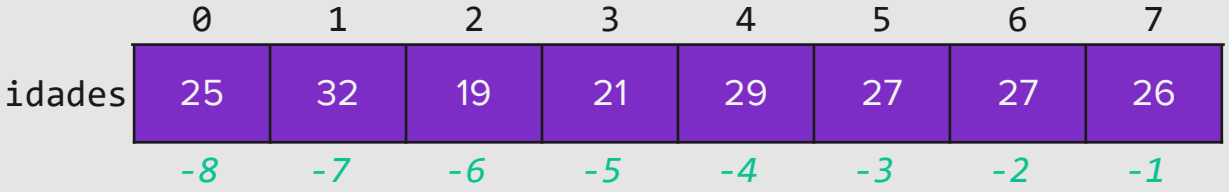
Imprimir na tela a idade da primeira pessoa da lista:

```
print(idades[0])
```

Calcular a média das 3 primeiras idades da lista:

```
media = (idades[0] + idades[1] + idades[2]) / 3
```

Acessando Elementos da Lista



Cuidado! O acesso a elementos indefinidos gera erros:

```
print(idades[8])
```

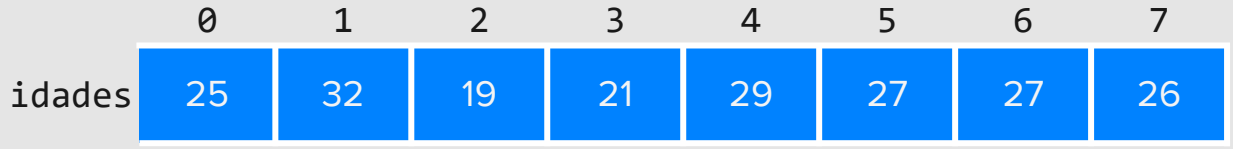
```
>> idades = [25, 32, 19, 21, 29, 27, 27, 36]
>> print(idades[8])
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(idades[8])
IndexError: list index out of range
```

Atualizando elementos

Adicionar a idade que está no índice 2 da lista:

```
idades[2] = 57
```

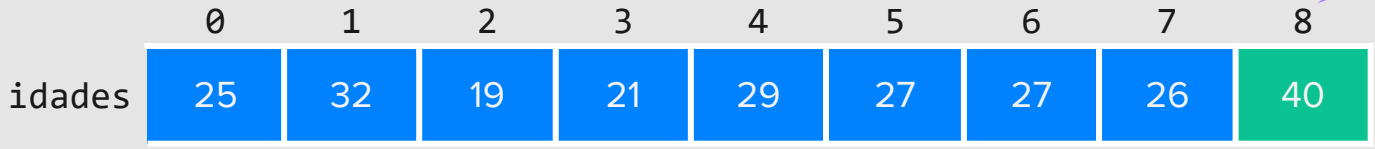
Inserção de Elementos



Adicionar uma idade (40) no final da lista:

```
idades.append(40)
```

Um **novo índice** é criado para referenciar o novo elemento.

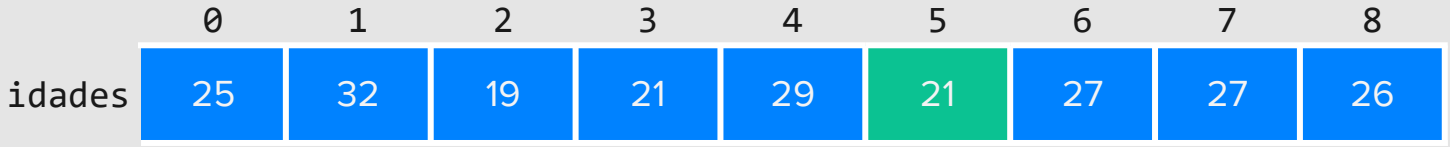


Inserção de Elementos

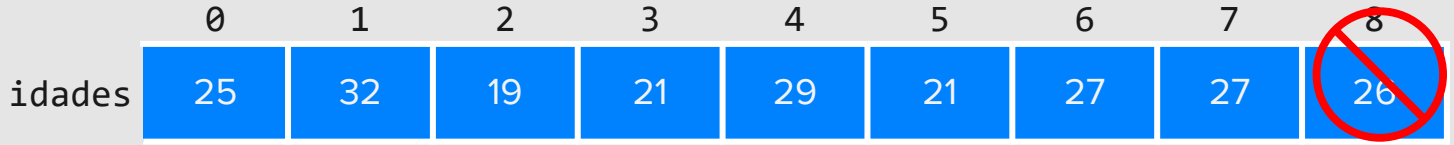
Adicionar uma idade (21) no índice 5 da lista:

```
idades.insert(5, 21)
```

Os demais elementos serão "**deslocados**" e terão novos índices.



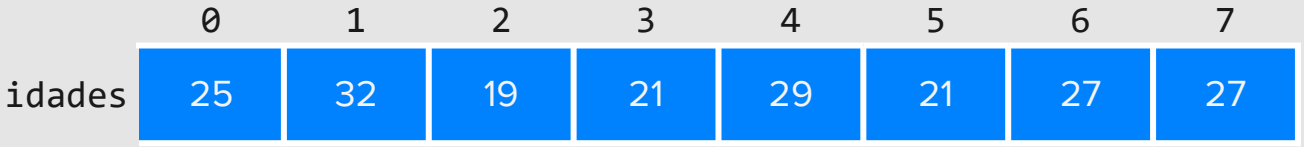
Remoção de Elementos



Remover o último elemento da lista:

```
idade_removida = idades.pop()
```

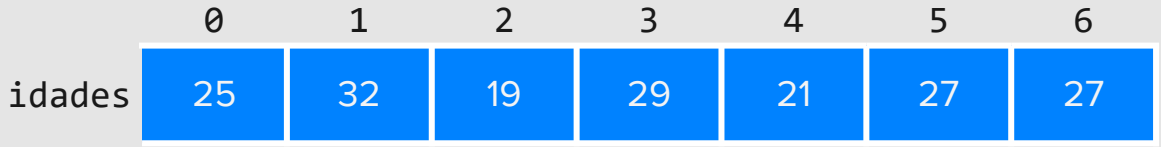
O **índice** e o **elemento** deixam de existir. O valor do elemento é **retornado**.



Remoção de Elementos

Remover o elemento de índice 3 da lista:

```
idade_removida = idades.pop(3)
```



Os demais elementos serão "**deslocados**" e terão novos índices. O índice que deixa de existir será o **último**.

Iterando sobre listas

Iteração com `while`:

```
idades = [25, 32, 19, 21, 29, 27, 27, 26]

i = 0
while i < 8:
    print(f"Idade {idades[i]} na posição {i}")
    i += 1
```

```
Idade 25 na posição 0
Idade 32 na posição 1
Idade 19 na posição 2
Idade 21 na posição 3
Idade 29 na posição 4
Idade 27 na posição 5
Idade 27 na posição 6
Idade 26 na posição 7
```

Com **`while`** é preciso manipular explicitamente os valores dos índices da lista usando, por exemplo, a própria variável (**i**) de controle do laço.

Iterando sobre listas

Iteração com `for`:

```
idades = [25, 32, 19, 21, 29, 27, 27, 26]

for item in idades:
    print(f"Idade {item}")
```

Com **for** o controle da repetição se dá conforme tamanho (quantidade de elementos) da lista. Dentro do bloco da repetição a variável **item** assumirá cada um dos valores contidos na lista **idades**.

```
Idade 25
Idade 32
Idade 19
Idade 21
Idade 29
Idade 27
Idade 27
Idade 26
```

Iterando sobre listas

Podemos utilizar a função enumerate para obter os índices de cada posição da lista

```
idades = [25, 32, 19, 21, 29, 27, 27, 26]

for indice, item in enumerate(idades):
    print(f"Posição [{indice}] Idade {item}")
```

```
Posição [0] Idade 25
Posição [1] Idade 32
Posição [2] Idade 19
Posição [3] Idade 21
Posição [4] Idade 29
Posição [5] Idade 27
Posição [6] Idade 27
Posição [7] Idade 26
```

Retornando ao Problema 0

Enunciado de um problema:

Ler as notas finais de 30 alunos. Calcular e informar a média da turma. Informar as notas que são superiores à média calculada.

```
ALUNOS = 30

# Inicializa a lista vazia de notas
notas = []
soma = 0

# Obtém a nota de cada aluno
for aluno in range(ALUNOS):
    nota = float(input(f"Informe a nota do aluno {aluno + 1}: "))
    notas.append(nota)
    soma += nota

# Calcula a média
media = soma / ALUNOS

# Escreve a média
print(f"Media da turma: {media:.2f}")

# Escreve as notas acima da média
for nota in notas:
    if nota > media:
        print(nota)
```

A lista notas inicia vazia

Adicionamos as notas na lista com append

Comparamos as notas com a média para decidir quais devemos imprimir

Importante Lembrar

* O índice da primeira posição de uma lista é sempre zero
  - → Por exemplo, é preciso cuidar porque
    - O elemento da **terceira** posição de uma lista tem o **índice 2**
    - Em uma lista com **10** elementos o **último índice positivo válido é 9**
    - Índices **negativos** são úteis para acessar a lista de trás para frente

* Acessar índices inexistentes gera erros do tipo `IndexError`

* Os índices são sempre inteiros e consecutivos
  - → Portanto, **não** é possível declarar uma lista com:
    - Apenas índices pares
    - Índices com valores reais
    - Texto como índices

Exercício 1

* Escreva um programa em Python que:
  1. Preencha um arranjo de números inteiros com 20000 valores aleatórios
     - Os valores devem ser números aleatórios entre 1000 e 100000.
     - Utilize as funções da biblioteca `random`
  2. Procure o maior número
     - Informe o valor e a posição
  3. Procure o menor número
     - Informe o valor e a posição
  4. Calcule o valor médio
     - Informe o valor médio
  5. Procure a posição que tem o valor mais próximo do valor médio
     - Informe o valor e a posição

Outras Operações sobre listas

* Remoção por conteúdo
* Tamanho da Lista
* Contar quantidade de ocorrências de um elemento
* Ordenar a lista
* Limpar (esvaziar) uma lista

Remoção por Conteúdo

* É possível localizar e remover um elemento de uma lista de várias formas. Uma das mais práticas é através do método remove do objeto list:

```
lista.remove(x)
```

* **Remove o primeiro** item encontrado na lista (lista) cujo valor é igual a x. Se não existir valor igual, uma exceção `ValueError` é lançada.
* Para evitar o `ValueError`, podemos tanto usar um bloco `try-except` quanto garantir que o elemento existe na lista usando o operador `in`, por exemplo:

```
if x in lista:
    lista.remove(x)
    print("Item removido!")
else:
    print("Item não encontrado!")
```

O mesmo operador `in` é utilizado tanto para percorrer uma lista com `for` quanto para verificar a existência de um valor qualquer com `if`.

Tamanho de uma Lista

* A função embutida `len` **devolve o comprimento** (o número de itens) de uma lista ou outros objetos que discutiremos mais adiante neste curso.

```
tamanho = len(lista)
```

* **Exemplo**: Podemos usar a função len para garantir que um valor de índice informado pelo usuário esteja no intervalo válido.

```
n = int(input("Informe o número do item para remover: "))
if n >= 0 and n < len(lista):
    lista.pop(n)
    print("Item removido!")
else:
    print("Índice Inválido!")
```

Um valor n informado pelo usuário é válido se estiver entre `0` e `len(lista) - 1`, inclusive.

Contagem de Elementos

* O método count **devolve** o número de vezes em que o elemento x aparece na lista

```
lista.count(x)
```

* **Exemplo**: Podemos utilizar a função count para remover itens duplicados de uma lista

```
lista = [1, 2, 3, 4, 4, 3, 2, 1]
for x in lista:
    if lista.count(x) > 1:
        lista.remove(x)
        print(f"{x} removido!")

print(lista)
```

A lógica do código a seguir é: para todo x pertencente à lista, se existe mais de uma ocorrência de x em `lista` (*i.e.*, `count` devolve 2 ou mais), então retira-se a primeira ocorrência de x com `remove`.

Uma alternativa para resolver esse problema seria evitar a inserção de itens duplicados. Você consegue pensar em como fazer isso?

Ordenando Listas

* Ordenar conjuntos de dados é uma operação relativamente complicada. Para nossa sorte, listas em Python tornam esse procedimento bem fácil através do método `sort`:

```
lista.sort(reverse = True)
```

* Ordena os itens na lista. O argumento `reverse` define se a ordenação será crescente ou decrescente. Esse método é baseado na função embutida `sorted` e fornece mais opções de ordenação que serão exploradas mais adiante neste curso.

Ordenando Listas

* O código a seguir trata das duas opções, crescente (c) ou decrescente (d), e faz a chamada apropriada do método sort:

```
lista = [12, 7, 97, -4, 3, 78, 14, 94, -11]

op = input("Ordem crescente (c) ou decrescente (d)? ")

if op == "c":
    lista.sort()
    print(lista)
elif op == "d":
    lista.sort(reverse=True)
    print(lista)
else:
    print("Opção Inválida!")
```

O método `list.reverse()` também poderia ser usado para inverter a ordem dos elementos na lista (independente da sua ordem).

Limpando a lista

* Limpar uma lista é uma operação bastante simples, para isso dispomos do método `clear` do objeto `list`:

```
lista.clear()
```

Problema 01: Lista de Compras

Enunciado do problema:

Um belo dia você acorda e percebe que não tem mais nada na sua geladeira. Como você é um programador organizado, você resolve fazer um programa para controlar a sua **lista de compras** do mercado com as seguintes operações:

1. inserir um item (tipo str) na lista
2. remover um item (pelo conteúdo ou pela posição)
3. remover itens duplicados
4. ordenar a lista em ordem alfabética (crescente ou decrescente)
5. imprimir a lista de compras
6. limpar a lista (remover tudo)