Apostila Completa: O Universo das Listas em Python

Olá! Prepare-se para mergulhar em um dos conceitos mais importantes da programação: as **listas**. Elas são a base para organizar dados de forma eficiente, e ao final desta apostila, você terá todo o conhecimento para usá-las como um profissional.

Capítulo 1: O Que É uma Lista?

Imagine que você tem uma gaveta com várias divisórias numeradas. Você pode colocar um objeto em cada divisória. Uma **lista** em Python é exatamente essa gaveta: uma coleção de itens guardados em uma ordem específica.

As listas têm três características principais:

1. **Ordem fixa**: A ordem em que você adiciona os itens é mantida.
2. **Mutável (editável)**: Você pode adicionar, remover ou modificar os itens a qualquer momento.
3. **Flexível**: Você pode guardar diferentes tipos de dados na mesma lista, como textos, números, ou até outras listas.

Como Criar uma Lista

Para criar uma lista, basta usar colchetes [] e separar os itens com vírgulas.

Python

```
# Uma lista de nomes (todos do tipo string)
nomes = ["João", "Maria", "Pedro"]

# Uma lista de números
precos = [10.50, 20.00, 3.75]

# Uma lista que mistura tipos (string, int, float)
produtos_e_precos = ["caneta", 2, 1.50, "mochila", 1, 120.00]

# Uma lista vazia (ótima para começar a construir)
lista_vazia = []
```

Capítulo 2: Acessando e Modificando Itens

Para pegar ou mudar um item de dentro da sua lista, você precisa usar a posição dele, que chamamos de **índice**.

Entendendo os Índices (A Posição do Item)

A regra mais importante: a contagem dos índices **sempre começa do zero**.

Item	"maçã"	"banana"	"uva"	"morango"
Índice	**0**	**1**	**2**	**3**

Você também pode contar de trás para frente usando índices negativos. O índice **-1** é o último item, **-2** é o penúltimo, e assim por diante.

Python

```
frutas = ["maçã", "banana", "uva", "morango"]

# Acessando o primeiro item (o da posição 0)
print(frutas[0]) # Saída: maçã

# Acessando o último item (usando o índice negativo)
print(frutas[-1]) # Saída: morango
```

Modificando Itens

Para trocar o valor de um item, você simplesmente o acessa pelo índice e dá a ele um novo valor.

Python

```
lista_de_compras = ["pão", "leite", "arroz"]

# Ops, vamos trocar "arroz" por "ovos"
lista_de_compras[2] = "ovos"

print(lista_de_compras) # Saída: ['pão', 'leite', 'ovos']
```

Pegando um Pedaço (Slicing)

O **slicing** (fatiar) permite que você pegue um ou mais itens de uma só vez, usando a sintaxe [início:fim].

Atenção: O item no índice de **fim** não é incluído no resultado.

Python

```
numeros = [0, 1, 2, 3, 4, 5, 6]

# Pegando os itens do índice 2 até o 4
sub_lista = numeros[2:5]
print(sub_lista) # Saída: [2, 3, 4]

# Pegando do início até o índice 3
primeiros_itens = numeros[:4]
print(primeiros_itens) # Saída: [0, 1, 2, 3]

# Pegando do índice 4 até o final
resto_da_lista = numeros[4:]
print(resto_da_lista) # Saída: [4, 5, 6]
```

Capítulo 3: Os Métodos da Lista (As Ações)

Métodos são como "botões" que a sua lista tem para realizar ações.

Adicionando Itens

- **.append(item)**: Adiciona um item no **final** da lista.
- **.insert(posicao, item)**: Adiciona um item em uma **posição específica**.

Python

```
minha_lista = ["A", "B", "C"]

minha_lista.append("D")
print(f"Depois de append: {minha_lista}") # Saída: ['A', 'B', 'C', 'D']

minha_lista.insert(1, "X")
print(f"Depois de insert: {minha_lista}") # Saída: ['A', 'X', 'B', 'C', 'D']
```

Removendo Itens

- **.remove(item)**: Remove a **primeira ocorrência** do item pelo **valor**.
- **.pop(posicao)**: Remove o item pela **posição**. Este método é útil porque ele te **devolve** o item que foi removido.

Python

```
numeros = [10, 20, 30, 20]

numeros.remove(20)
print(f"Depois de remove: {numeros}") # Saída: [10, 30, 20]

item_retirado = numeros.pop(1)
print(f"O item retirado com pop foi: {item_retirado}") # Saída: 30
print(f"Lista depois de pop: {numeros}") # Saída: [10, 20]
```

Ordenando e Invertendo

- **.sort()**: Organiza a lista em ordem crescente (numérica ou alfabética).
- **.sort(reverse=True)**: Organiza em ordem decrescente.
- **.reverse()**: Inverte a ordem atual da lista.

Atenção: Estes métodos **modificam a lista original**.

Python

```
numeros = [3, 1, 4, 2]
numeros.sort()
print(f"Lista ordenada: {numeros}") # Saída: [1, 2, 3, 4]

letras = ["c", "b", "a"]
letras.reverse()
print(f"Lista invertida: {letras}") # Saída: ['a', 'b', 'c']
```

Capítulo 4: List Comprehension (O Atalho Mágico)

A **List Comprehension** é a forma elegante e rápida de criar novas listas. É um atalho para um laço for que gera uma lista.

Problema: Crie uma lista com o dobro de cada número de 1 a 5.

Forma tradicional (mais longa):

Python

```
dobros = []
for numero in range(1, 6):
    dobros.append(numero * 2)
print(dobros) # Saída: [2, 4, 6, 8, 10]
```

Com List Comprehension (uma linha!):
A sintaxe é: [o_que_eu_quero_fazer for item in lista_original if uma_condicao]

Python

```
dobros = [numero * 2 for numero in range(1, 6)]
print(dobros) # Saída: [2, 4, 6, 8, 10]
```

Você pode incluir uma condição if para filtrar itens:

Python

```
# Pegando apenas os números pares de 0 a 9
pares = [x for x in range(10) if x % 2 == 0]
print(pares) # Saída: [0, 2, 4, 6, 8]
```

Capítulo 5: Pilhas e Filas (Aplicações Práticas)

As listas são tão úteis que podem simular duas estruturas de dados essenciais.

A Pilha (LIFO): O Princípio da Pilha de Pratos

LIFO significa **L**ast **I**n, **F**irst **O**ut (Último a Entrar, Primeiro a Sair). Você sempre adiciona e remove do topo.

- **Para adicionar (push)**: Use lista.append()
- **Para remover (pop)**: Use lista.pop() (sem nenhum valor).

Python

```
pilha = []
pilha.append("prato 1")
pilha.append("prato 2")
prato_retirado = pilha.pop()

print(f"O último prato retirado foi: {prato_retirado}")
print(f"Pilha restante: {pilha}")
# Saída:
# O último prato retirado foi: prato 2
# Pilha restante: ['prato 1']
```

A Fila (FIFO): O Princípio da Fila do Banco

FIFO significa **F**irst **I**n, **F**irst **O**ut (Primeiro a Entrar, Primeiro a Sair). Você adiciona no final e remove do início.

- **Para adicionar (enqueue)**: Use lista.append()
- **Para remover (dequeue)**: Use lista.pop(0) (removendo o item do índice 0).

Python

```
fila = []
fila.append("João")
fila.append("Maria")
proximo_atendido = fila.pop(0)

print(f"O primeiro cliente atendido foi: {proximo_atendido}")
print(f"Fila restante: {fila}")
# Saída:
# O primeiro cliente atendido foi: João
# Fila restante: ['Maria']
```

Capítulo 6: Desafios de Lógica (Algoritmos Complexos)

Vamos aplicar tudo o que aprendemos para resolver dois problemas que parecem complexos, mas que se resolvem com a lógica das listas.

Desafio 1: Encontrar e Contar Itens Únicos

Problema: Dada uma lista com nomes repetidos, crie uma nova lista apenas com os nomes que aparecem uma única vez.

Lógica:

1. Crie uma lista vazia para os nomes únicos.
2. Percorra a lista original com um for.
3. Para cada nome, verifique se ele **já existe** na lista de nomes únicos.
4. Se não existir, adicione-o.

Python

```
nomes = ["João", "Maria", "Pedro", "João", "Ana", "Maria"]
unicos = []

for nome in nomes:
    # O "not in" verifica se o item não está na lista
    if nome not in unicos:
        unicos.append(nome)

print(unicos) # Saída: ['João', 'Maria', 'Pedro', 'Ana']
```

Desafio 2: Simular um Sorteio sem Repetição

Problema: Dado uma lista de nomes, sorteie três nomes diferentes. Um nome não pode ser sorteado mais de uma vez. Para isso, vamos precisar do módulo random.

Lógica:

1. Crie uma lista com os participantes e uma lista vazia para os vencedores.
2. Repita o processo 3 vezes:
3. Escolha um nome aleatório da lista de participantes.
4. Adicione esse nome à lista de vencedores.
5. Remova esse nome da lista de participantes para que ele não possa ser escolhido novamente.

Python

```
import random

participantes = ["Ana", "Bruno", "Carlos", "Daniel", "Eva"]
vencedores = []

print("Realizando o sorteio...")

# O laço roda 3 vezes para escolher 3 vencedores
for _ in range(3):
    # Escolhe um nome aleatório
    sorteado = random.choice(participantes)

    # Adiciona o nome à lista de vencedores
    vencedores.append(sorteado)

    # Remove o nome da lista original
    participantes.remove(sorteado)

print(f"\nOs três vencedores são: {vencedores}")
# O resultado vai ser diferente a cada vez que você rodar!
```