Estruturas de Dados em Python

1 Listas

Definição: Uma lista é uma coleção ordenada e mutável de elementos. Ela permite duplicatas, sendo ideal para armazenar diversos itens.

1.1 Principais operações

- Adicionar elementos: `append()` ou `extend()`
- Remover elementos: `remove()` ou `pop()`
- Ordenar: `sort()`
- Acessar elementos: através de índices

1.2 Exemplo

```
# Criando uma lista
frutas = ["maçã", "banana", "laranja"]
print("Lista original:", frutas)

# Adicionando elementos
frutas.append("uva")
print("Após append:", frutas)

# Removendo um elemento
frutas.remove("banana")
print("Após remove:", frutas)

# Acessando o primeiro elemento
print("Primeiro elemento:", frutas[0])

```

```python
# Ordenando a lista
frutas.sort()
print("Após sort:", frutas)
```

1.3 Exercícios

1. Crie uma lista com números de 1 a 10. Remova o número 5 e adicione o número 15.
2. Ordene uma lista de nomes em ordem alfabética.
3. Acesse e exiba o terceiro elemento de uma lista.

2 Tuplas

Definição: Uma tupla é uma coleção ordenada e imutável de elementos. Por ser imutável, seus valores não podem ser alterados após a criação.

2.1 Principais operações

- Acessar elementos: através de índices
- Contar elementos: `count()`
- Encontrar índice: `index()`

2.2 Exemplo

```python
# Criando uma tupla
cores = ("vermelho", "azul", "verde")
print("Tupla original:", cores)

# Acessando elementos
print("Primeiro elemento:", cores[0])

# Contando elementos
print("Contagem de 'azul':", cores.count("azul"))

# Encontrando o índice de um elemento
print(" ndice  de 'verde':", cores.index("verde"))
```

2.3 Operações com Tuplas

Indexação

Assim como nas listas, podemos acessar elementos de uma tupla usando índices.

```
tupla = (10, 20, 30)
print("Primeiro elemento:", tupla[0])  # Resultado: 10
print("Último elemento:", tupla[-1])  # Resultado: 30
```

Fatiamento (Slicing)

Podemos extrair partes da tupla utilizando a notação de start:stop:step.

```
tupla = (1, 2, 3, 4, 5)
print("Elementos do índice 1 ao 3:", tupla[1:4])  # Resultado: (2, 3, 4)
print("Elementos com passo 2:", tupla[::2])       # Resultado: (1, 3, 5)
```

Concatenação

Duas ou mais tuplas podem ser concatenadas usando o operador +.

```
tupla1 = (1, 2)
tupla2 = (3, 4)
tupla3 = tupla1 + tupla2
print("Tupla concatenada:", tupla3)  # Resultado: (1, 2, 3, 4)
```

Repetição

Tuplas podem ser repetidas utilizando o operador *.

```
tupla = (1, 2)
print("Tupla repetida:", tupla * 3)  # Resultado: (1, 2, 1, 2, 1, 2)
```

Pertinência (Membership)

Podemos verificar se um elemento está presente na tupla com os operadores `in` e `not in`.

```
tupla = (1, 2, 3)
print(2 in tupla)   # Resultado: True
print(5 not in tupla)  # Resultado: True
```

Contar Elementos

O método `count()` pode ser usado para contar quantas vezes um valor aparece na tupla.

```python
tupla = (1, 2, 2, 3, 4, 2)
print("Quantidade de '2':", tupla.count(2))  # Resultado: 3
```

Encontrar Índices

O método `index()` retorna o índice da primeira ocorrência de um elemento na tupla.

```python
tupla = (10, 20, 30, 40)
print(" ndice  de 30:", tupla.index(30))  # Resultado: 2
```

Tamanho da Tupla

A função `len()` retorna o número de elementos na tupla.

```python
tupla = (1, 2, 3)
print("Tamanho da tupla:", len(tupla))  # Resultado: 3
```

Iteração

Podemos percorrer os elementos de uma tupla com um laço `for`.

```python
tupla = ("a", "b", "c")
for elemento in tupla:
    print("Elemento:", elemento)
```

2.4 Exercícios

1. Crie uma tupla com cinco números e acesse o último elemento.
2. Conte quantas vezes o número 3 aparece em uma tupla.
3. Encontre o índice do valor "Python"em uma tupla.

3 Sets (Conjuntos)

Definição: Um set é uma coleção não ordenada e que não possui elementos duplicados. Ele é mutável, mas seus elementos devem ser imutáveis.

3.1 Principais operações

- Adicionar elementos: `add()`
- Remover elementos: `remove()` ou `discard()`

- Operações de conjuntos: união (`union()`), interseção (`intersection()`), diferença (`difference()`)

3.2 Exemplo

```python
# Criando um set
numeros = {1, 2, 3}
print("Set original:", numeros)

# Adicionando elementos
numeros.add(4)
print("Após add:", numeros)

# Removendo elementos
numeros.remove(2)
print("Após remove:", numeros)

# Operações de conjuntos
pares = {2, 4, 6}
print("União:", numeros.union(pares))
print("Interseção:", numeros.intersection(pares))
print("Diferença:", numeros.difference(pares))
```

3.3 Exercícios

1. Crie um set com os números 1, 2, 3 e adicione o número 4.
2. Faça a interseção entre dois sets: {1, 2, 3} e {2, 3, 4}.
3. Remova um elemento de um set e exiba o resultado.

Estilos de formatação do comando print()

O comando `print()` é usado para exibir valores e mensagens no console. Python oferece três estilos principais para formatar e printar variáveis: concatenação, formatação tradicional e f-strings.

5.1. Concatenar Strings com +

Neste método, usamos o operador `+` para concatenar textos e variáveis convertidas para strings.

Exemplo:

```
nome = "Maria"
idade = 25
print("Meu nome é " + nome + " e eu tenho " + str(idade) + "
    anos.")
```

Limitação: Precisamos converter variáveis não-textuais (como números) para strings usando `str()`.

5.2. Formatação Tradicional com `format()`

Utilizamos o método `.format()` para substituir marcadores {} pelos valores das variáveis.

Exemplo:

```
nome = "Maria"
idade = 25
print("Meu nome é {} e eu tenho {} anos.".format(nome, idade)
    )
```

Vantagens:

- Permite reutilizar variáveis na string.
- Pode especificar a ordem dos parâmetros: `{1}` `{0}`.

Exemplo com indexação:

```
print("Idade: {1}, Nome: {0}".format(nome, idade))
```

5.3. Formatação com F-Strings (Python 3.6+)

Com f-strings, colocamos o `f` antes da string e inserimos as variáveis diretamente entre {}.

Exemplo:

```
nome = "Maria"
idade = 25
print(f"Meu nome é {nome} e eu tenho {idade} anos.")
```

Vantagens:

- Mais legível e intuitivo.

- Permite operações diretas dentro das chaves {}.

Exemplo com operação:

```
print(f"Daqui a 5 anos, terei {idade + 5} anos.")
```

Exercícios de Estilos de Print

1. Crie uma mensagem concatenando variáveis com o operador `+`.
2. Use `.format()` para exibir o nome e a profissão de uma pessoa.
3. Crie uma frase usando f-strings para mostrar uma operação matemática com variáveis.

4 Laços de Repetição

Os laços de repetição são usados para executar um bloco de código várias vezes, de acordo com uma condição.

4.1 For

Definição: O laço `for` é usado para iterar sobre uma sequência (como lista, tupla, ou string) ou um intervalo definido.

Quando usar: Utilize quando souber o número exato de iterações.

Exemplo:

```
# Iterando sobre uma lista
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(f"Fruta: {fruta}")

# Iterando com range()
for i in range(5):  # 0 a 4
    print(f"Iteração {i}")
```

Exercícios:

- Itere sobre uma lista de números e exiba apenas os números pares.
- Use o `range()` para exibir os números de 1 a 10.
- Itere sobre as letras de uma palavra e exiba cada letra.

4.2 While

Definição: O laço `while` repete um bloco de código enquanto uma condição for verdadeira.

Quando usar: Utilize quando não souber o número exato de iterações, mas houver uma condição.

Exemplo:

```
# Contando até 5
contador = 1
while contador <= 5:
    print(f"Contador: {contador}")
    contador += 1  # Incremento
```

Exercícios:

- Use um `while` para contar de 1 a 10.
- Utilize um `while` para exibir os números enquanto eles forem menores que 20.
- Crie um programa que pare de executar quando o usuário digitar "sair".

4.3 Do-While

Definição: Python não possui diretamente um `do-while`, mas podemos simulá-lo com `while` para garantir que o código execute pelo menos uma vez.

Quando usar: Utilize quando precisar executar o bloco de código ao menos uma vez antes de verificar a condição.

Exemplo:

```
# Simulação de do-while
valor = 0
while True:
    valor += 1
    print(f"Valor: {valor}")
    if valor >= 5:  # Condição de saída
        break
```

Exercícios:

- Simule um do-while para pedir que o usuário insira um número e pare quando ele digitar "0".

- Crie um programa que exiba números aleatórios enquanto eles forem menores que 50.
- Use uma estrutura semelhante a `do-while` para validar uma entrada de texto.

Exercícios:

- Simule um `switch` para retornar o nome do dia da semana com base no número (1 a 7).
- Crie um dicionário de operações matemáticas (+, -, *, /) e execute a operação desejada.
- Crie um programa que simule `switch` para retornar a classificação de uma nota (A, B, C, etc.).

5 Condicionais (If, Else, Elif)

Definição: Estruturas condicionais permitem executar diferentes blocos de código com base em condições.

If: Executa um bloco de código se a condição for verdadeira.
Else: Executa um bloco alternativo se a condição no `if` for falsa.
Elif: Adiciona outras condições além do `if`.
Exemplo:

```
idade = 20

# Estrutura condicional
if idade < 18:
    print("Menor de idade.")
elif idade >= 18 and idade < 65:
    print("Adulto.")
else:
    print("Idoso.")
```

Exercícios:

- Escreva um programa que exiba "Aprovado"se a nota for maior que 7, e "Reprovado"caso contrário.
- Crie um programa que classifique a idade de uma pessoa (criança, adolescente, adulto, idoso).

- Use `elif` para criar um sistema de categorias baseado na altura de uma pessoa.

6 O que é o range() no Python?

Definição: O `range()` é uma função embutida em Python que gera uma sequência de números. Ele é comumente usado em loops, como o `for`, para iterar por uma quantidade definida de vezes.

7 Por que usar o range()?

- Para controlar o início, o fim e o passo de uma sequência numérica.
- Para evitar a necessidade de criar manualmente listas grandes de números.
- Para iterar eficientemente em números sem precisar armazená-los na memória.

8 Como funciona o range()?

A função `range()` pode ser usada de três formas principais:

8.1 `range(stop)`

Gera uma sequência de números que começa do zero e vai até `stop - 1`.

Exemplo:

```
for i in range(5):  # De 0 até 4
    print(i)
```

8.2 `range(start, stop)`

Gera uma sequência que começa em `start` e vai até `stop - 1`.

Exemplo:

```
for i in range(1, 5):  # De 1 até 4
    print(i)
```

8.3 `range(start, stop, step)`

Gera uma sequência começando em `start`, indo até `stop - 1`, com intervalos definidos pelo `step`.

Exemplo:

```
for i in range(0, 10, 2):  # De 0 até 8 com passo de 2
    print(i)
```

9 Característica importante

O `range()` não gera uma lista diretamente, mas sim um objeto iterável que é criado de forma eficiente. Se você quiser transformá-lo em uma lista, pode usar `list()`.

Exemplo:

```
lista = list(range(5))  # Converte o range em uma lista
print(lista)  # [0, 1, 2, 3, 4]
```

10 Exercícios para prática

- Use o `range()` para imprimir os números de 0 a 9.
- Crie uma sequência de números pares de 2 a 20 usando o `range()`.
- Use `range()` para imprimir números de 10 a 1 em ordem decrescente.