Strings, Coleções e Padrões em Python

Informações do Autor

- **Nome:** ANIMA Lab
- **Data de atualização**: 30/09/2025

1) Strings e suas operações

Uma string é uma sequência de caracteres que pode representar palavras, frases, números escritos como texto ou até símbolos. No Python, escrevemos uma string entre aspas simples (`'texto'`) ou aspas duplas (`"texto"`). O computador armazena cada caractere com uma posição numérica chamada índice, que começa em `0`.

Indexação

A indexação é o acesso direto a um caractere da string.

- O primeiro caractere está na posição `0`.
- O último caractere pode ser acessado com índices negativos, como `-1`.

```
palavra = "Python"
print(palavra[0])   # P (primeira letra)
print(palavra[-1])  # n (última letra)
```

Fatiamento

Podemos pegar uma parte da string usando a notação `[início:fim]`. O índice de início é incluído, e o índice de fim é excluído.

```
palavra = "Python"
print(palavra[0:2])   # Py (pega da posição 0 até a 1)
print(palavra[2:5])   # tho (pega da posição 2 até a 4)
print(palavra[:4])    # Pyth (início até posição 3)
print(palavra[3:])    # hon (da posição 3 até o fim)
```

Concatenação

Podemos juntar strings com o operador +.

```
nome = "Maria"
sobrenome = "Silva"
print(nome + " " + sobrenome)  # Maria Silva
```

Comparação

Strings podem ser comparadas com operadores relacionais:

- == (igualdade)
- != (diferente)
- < e > (ordem alfabética, baseada na tabela ASCII)

```
print("casa" == "casa")   # True
print("casa" != "carro")  # True
print("bola" < "casa")    # True (porque "b" vem antes de "c")
```

Imutabilidade

Strings em Python são imutáveis. Isso significa que, uma vez criada, não é possível alterar uma parte da string diretamente. Se quisermos mudar um caractere, precisamos criar uma nova string.

```
palavra = "Python"
# palavra[0] = "J"  # Isso gera erro

nova = "J" + palavra[1:]
print(nova)  # Jython
```

Exercício-1 – Strings

Um centro de convivência de idosos deseja gerar crachás automáticos. Crie um programa que:

1. Receba o nome completo de um participante.
2. Mostre apenas as iniciais.
3. Exiba também o primeiro nome em letras maiúsculas.

---

2) Coleções Básicas (*built-in*) em Python

Em muitos problemas, precisamos guardar vários valores juntos. Por exemplo: uma lista de nomes de idosos, as coordenadas de um ponto ou os dados de um participante (nome, idade e cidade). Para isso, o Python oferece coleções que já vêm prontas na linguagem.

As coleções mais comuns são: listas, tuplas e dicionários.

Listas (`list`)

Uma lista é uma sequência de elementos que pode mudar depois de criada. Isso significa que podemos adicionar, remover e alterar itens.

- Criamos listas com colchetes `[]`.
- Podemos misturar diferentes tipos de dados na mesma lista (embora, em ciência de dados, geralmente mantemos um tipo só para facilitar).
- Cada elemento tem uma posição numérica chamada índice, que começa em `0`.

Exemplo:

```
numeros = [5, 2, 9]   # cria uma lista com três elementos
numeros.append(4)     # adiciona o número 4 no final
numeros.sort()        # coloca os números em ordem crescente
print(numeros)        # [2, 4, 5, 9]
```

Operações comuns:

- `lista[i]` → acessa o elemento na posição `i`.
- `lista.append(x)` → adiciona um elemento no final.
- `lista.remove(x)` → remove o primeiro valor igual a `x`.
- `lista.sort()` → organiza os valores em ordem.

Tuplas (`tuple`)

Uma tupla parece uma lista, mas não pode ser modificada depois de criada. Por isso, dizemos que é imutável.

- Criamos tuplas com parênteses `()`.
- São usadas quando queremos garantir que os dados não mudem.
- Muito comuns para representar pares de valores, como coordenadas.

Exemplo:

```
coord = (10, 20)
print(coord[0])   # 10
print(coord[1])   # 20
```

Se tentarmos `coord[0] = 50`, o Python dará erro porque a tupla não pode ser alterada.

Dicionários (`dict`)

Um dicionário guarda pares no formato chave:valor. Diferente das listas e tuplas (acessadas por posição), no dicionário acessamos os valores pelas chaves.

- Criamos dicionários com chaves `{}`.
- A chave pode ser um texto, número ou outro tipo imutável.
- O valor pode ser qualquer tipo de dado.

Exemplo:

```
idoso = {"nome": "Maria", "idade": 72}
print(idoso["nome"])        # Maria
idoso["cidade"] = "Goiânia" # adiciona uma nova chave
print(idoso)
```

Saída:

```
{'nome': 'Maria', 'idade': 72, 'cidade': 'Goiânia'}
```

Operações comuns:

- `dicionario[chave]` → acessa o valor da chave.
- `dicionario[chave] = valor` → adiciona ou altera um par.
- `dicionario.keys()` → retorna todas as chaves.
- `dicionario.values()` → retorna todos os valores.

Exercício-2 – Coleções

Um pesquisador está organizando os dados de idosos atendidos em um centro de convivência. Ele precisa de um programa em Python que:

1. Armazene em uma lista de dicionários os dados de pelo menos 5 idosos, contendo nome, idade e cidade.
2. Mostre todos os registros cadastrados, um por linha.

3. Permita que o usuário digite um nome e o programa busque esse idoso, exibindo seus dados completos (ou informe que não foi encontrado).
4. Calcule e exiba a idade média dos idosos.
5. Descubra e mostre também:
    1. o nome do idoso mais velho;
    2. o nome do idoso mais novo.

Exercício-3 – Tupla

Um centro comunitário registrou informações fixas sobre a localização de diferentes unidades de atendimento a idosos. Cada unidade é representada por uma tupla no formato `(nome, latitude, longitude)`.

1. Crie uma tupla para cada uma das 3 unidades, com dados fictícios.
2. Mostre todas as tuplas criadas.
3. Acesse individualmente os dados da primeira unidade e mostre em uma frase, por exemplo:

    ```
    Unidade Central localizada em (latitude: -16.67, longitude: -
    ```

4. Explique no código, via comentários, por que escolhemos tuplas e não listas nesse caso.

Exercício-4 – Dicionário

Um sistema de cadastro precisa guardar dados de um único idoso em um dicionário, no formato:

```
idoso = {"nome": ..., "idade": ..., "cidade": ...}
```

Crie um programa que:

1. Preencha o dicionário com dados de um idoso.
2. Mostre todos os campos (`nome`, `idade` e `cidade`).
3. Permita atualizar a cidade para um novo valor digitado pelo usuário.
4. Adicione uma nova chave chamada `"telefone"` e preencha com um número fictício.
5. Mostre o dicionário atualizado.

---

3) Combinando padrões com coleções

Quando trabalhamos com coleções em Python (listas, tuplas, dicionários), muitas vezes queremos separar seus elementos em variáveis individuais. Esse processo é chamado de desempacotamento (*pattern matching*).

Em vez de acessar cada elemento por índice (`ponto[0]`, `ponto[1]`), podemos atribuir diretamente cada parte da coleção a uma variável.

Exemplo 1 – desempacotando uma tupla simples

```
ponto = (3, 4)

(x, y) = ponto
print("Coordenadas:", x, y)
```

Aqui, o valor `3` foi atribuído à variável `x`, e o valor `4` à variável `y`.

Exemplo 2 – desempacotando várias variáveis de uma vez

```
dados = ("Maria", 72, "Goiânia")

(nome, idade, cidade) = dados
print(nome, "-", idade, "anos -", cidade)
```

Saída:

```
Maria - 72 anos - Goiânia
```

Essa técnica deixa o código mais claro do que acessar cada elemento por índice (`dados[0]`, `dados[1]`, etc.).

Exemplo 3 – desempacotando estruturas aninhadas

Se uma coleção contém outras coleções dentro dela, também podemos desempacotar em vários níveis.

```
cor = ((255, 200, 100), "amarelo")

((r, g, b), nome) = cor
print("Cor:", nome, "| RGB:", r, g, b)
```

Aqui, a tupla `(255, 200, 100)` foi dividida em três variáveis (`r`, `g`, `b`), e a string `"amarelo"` foi atribuída à variável `nome`.

Exemplo 4 – ignorando valores com _

Se não precisamos de todos os elementos, podemos usar _ para ignorar um deles:

```
ponto = (10, 20, 30)

(x, _, z) = ponto
print(x, z)  # 10 30
```

Exercício-5 – Pattern Matching em registros de saúde

Um sistema de monitoramento coleta dados de vários idosos. Cada registro vem no formato:

```
((pressao_sistolica, pressao_diastolica), batimento)
```

Exemplo de lista de registros:

```
dados = [
    ((120, 80), 72),
    ((135, 90), 80),
    ((110, 70), 65),
    ((150, 95), 85)
]
```

Crie um programa que:

1. Percorra a lista de registros.
2. Use pattern matching (desempacotamento) para separar as variáveis `pressao_sistolica`, `pressao_diastolica` e `batimento`.
3. Mostre cada registro formatado, por exemplo:

   ```
   Pressão: 120x80 | Batimento: 72 bpm
   ```

4. Calcule e mostre a média dos batimentos cardíacos.
5. Identifique o registro com a maior pressão sistólica e exiba-o.

Exemplo esperado de saída

```
Pressão: 120x80 | Batimento: 72 bpm
Pressão: 135x90 | Batimento: 80 bpm
Pressão: 110x70 | Batimento: 65 bpm
Pressão: 150x95 | Batimento: 85 bpm

Média de batimentos: 75.5 bpm
Maior pressão sistólica registrada: 150x95 | Batimento: 85 bpm
```