Listas

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Entender o funcionamento da estrutura de dados lista.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Criação e acesso aos dados**

Etapa 2 Métodos da classe list

Etapa 1

Criação e acesso aos dados

Criando listas

Listas em Python podem armazenar de maneira sequencial qualquer tipo de objeto. Podemos criar listas utilizando o construtor **list,** a função range ou colocando valores separados por vírgula dentro de colchetes. Listas são objetos mutáveis, portanto podemos alterar seus valores após a criação.

Exemplo

```
frutas = ["laranja", "maca", "uva"]

frutas = []

letras = list("python")

numeros = list(range(10))

carro = ["Ferrari", "F8", 4200000, 2020, 2900, "São Paulo", True]
```

Acesso direto

A lista é uma sequência, portanto podemos acessar seus dados utilizando índices. Contamos o índice de determinada sequência a partir do zero.

Exemplo

```
frutas = ["maçã", "laranja", "uva", "pera"]
frutas[0]  # maçã
frutas[2]  # uva
```

Índices negativos

Sequências suportam indexação negativa. A contagem começa em -1.

Exemplo

```
frutas = ["maçã", "laranja", "uva", "pera"]
frutas[-1]  # pera
frutas[-3]  # laranja
```

Listas aninhadas

Listas podem armazenar todos os tipos de objetos Python, portanto podemos ter listas que armazenam outras listas. Com isso podemos criar estruturas bidimensionais (tabelas), e acessar informando os índices de linha e coluna.

Exemplo

```
matriz = [
    [1, "a", 2],
    ["b", 3, 4],
    [6, 5, "c"]
]

matriz[0]  # [1, "a", 2]
matriz[0][0]  # 1
matriz[0][-1]  # 2
matriz[-1][-1]  # "c"
```

Fatiamento

Além de acessar elementos diretamente, podemos extrair um conjunto de valores de uma sequência. Para isso basta passar o índice inicial e/ou final para acessar o conjunto. Podemos ainda informar quantas posições o cursor deve "pular" no acesso.

Exemplo

```
lista = ["p", "y", "t", "h", "o", "n"]

lista[2:]  # ["t", "h", "o", "n"]
lista[:2]  # ["p", "y"]
lista[1:3]  # ["y", "t"]
lista[0:3:2]  # ["p", "t"]
lista[::]  # ["p", "y", "t", "h", "o", "n"]
lista[::-1]  # ["n", "o", "h", "t", "y", "p"]
```

Iterar listas

A forma mais comum para percorrer os dados de uma lista é utilizando o comando **for**.

Exemplo

```
carros = ["gol", "celta", "palio"]

for carro in carros:
    print(carro)
```

Função enumerate

Às vezes é necessário saber qual o índice do objeto dentro do laço **for**. Para isso podemos usar a função **enumerate**.

Exemplo

```
carros = ["gol", "celta", "palio"]

for indice, carro in enumerate(carros):
    print(f"{indice}: {carro}")
```

Compreensão de listas

A compreensão de lista oferece uma sintaxe mais curta quando você deseja: criar uma nova lista com base nos valores de uma lista existente (filtro) ou gerar uma nova lista aplicando alguma modificação nos elementos de uma lista existente.

Filtro versão 1

```
numeros = [1, 30, 21, 2, 9, 65, 34]
pares = []

for numero in numeros:
    if numero % 2 == 0:
        pares.append(numero)
```

Filtro versão 2

```
numeros = [1, 30, 21, 2, 9, 65, 34]
pares = [numero for numero in numeros if numero % 2 == 0]
```

Modificando valores versão 1

```
numeros = [1, 30, 21, 2, 9, 65, 34]
quadrado = []

for numero in numeros:
    quadrado.append(numero ** 2)
```

Modificando valores versão 2

```
numeros = [1, 30, 21, 2, 9, 65, 34]
quadrado = [numero ** 2 for numero in numeros]
```

Percurso

~~Etapa 1~~ ~~Criação e acesso aos dados~~

Etapa 2 **Métodos da classe list**

Etapa 2

Métodos da classe list

[].append

```
lista = []

lista.append(1)
lista.append("Python")
lista.append([40, 30, 20])

print(lista)  # [1, "Python", [40, 30, 20]]
```

[].clear

```python
lista = [1, "Python", [40, 30, 20]]

print(lista)  # [1, "Python", [40, 30, 20]]

lista.clear()

print(lista)  # []
```

[].copy

```python
lista = [1, "Python", [40, 30, 20]]

lista.copy()

print(lista)  # [1, "Python", [40, 30, 20]]
```

[].count

```
cores = ["vermelho", "azul", "verde", "azul"]

cores.count("vermelho")  # 1
cores.count("azul")  # 2
cores.count("verde")  # 1
```

[].extend

```python
linguagens = ["python", "js", "c"]

print(linguagens)  # ["python", "js", "c"]

linguagens.extend(["java", "csharp"])

print(linguagens)  # ["python", "js", "c", "java", "csharp"]
```

[].index

```
linguagens = ["python", "js", "c", "java", "csharp"]

linguagens.index("java")  # 3
linguagens.index("python")  # 0
```

[].pop

```
linguagens = ["python", "js", "c", "java", "csharp"]

linguagens.pop()  # csharp
linguagens.pop()  # java
linguagens.pop()  # c
linguagens.pop(0)  # python
```

[].remove

```
linguagens = ["python", "js", "c", "java", "csharp"]

linguagens.remove("c")

print(linguagens)  # ["python", "js", "java", "csharp"]
```

[].reverse

```
linguagens = ["python", "js", "c", "java", "csharp"]

linguagens.reverse()

print(linguagens)  # ["csharp", "java", "c", "js", "python"]
```

[].sort

```
linguagens = ["python", "js", "c", "java", "csharp"]
linguagens.sort()  # ["c", "csharp", "java", "js", "python"]

linguagens = ["python", "js", "c", "java", "csharp"]
linguagens.sort(reverse=True)  # ["python", "js", "java", "csharp", "c"]

linguagens = ["python", "js", "c", "java", "csharp"]
linguagens.sort(key=lambda x: len(x))  # ["c", "js", "java", "python", "csharp"]

linguagens = ["python", "js", "c", "java", "csharp"]
linguagens.sort(key=lambda x: len(x), reverse=True)  # ["python", "csharp",
"java", "js", "c"]
```

len

```
linguagens = ["python", "js", "c", "java", "csharp"]

len(linguagens)  # 5
```

sorted

```
linguagens = ["python", "js", "c", "java", "csharp"]

sorted(linguagens, key=lambda x: len(x))  # ["c", "js", "java", "python",
"csharp"]

sorted(linguagens, key=lambda x: len(x), reverse=True)  # ["python", "csharp",
"java", "js", "c"]
```

Percurso

~~Etapa 1~~ ~~Criação e acesso aos dados~~

~~Etapa 2~~ ~~Métodos da classe list~~

Links Úteis

- https://github.com/digitalinnovationone/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)