

FIAP

# Python – Listas Tupla Dict Set

Prof. Dr. Luiz W Tavares  
2024

## Introdução ao tipo list

- Em Python, uma lista (list) é uma coleção ordenada de itens que pode conter elementos de diferentes tipos, como inteiros, strings, ou até outras listas.
- As listas são mutáveis, o que significa que os itens podem ser alterados após a criação da lista.



## List

### Criação de uma Lista

# Lista vazia

```
lista_vazia = []
```

# Lista com elementos

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

```
palavras = ['python', 'é', 'legal']
```

```
mista = [1, 'hello', 3.5, [1, 2, 3]]
```

## List

### Acessando Elementos

- Use índices para acessar elementos individuais.
- Índices começam em 0.

`primeiro_elemento = numeros[0] # Retorna 1`

`ultimo_elemento = numeros[-1] # Retorna 5`

## List

### Modificando Elementos

- Modifique um item diretamente pelo índice.

`numeros[0] = 10` # Muda o primeiro elemento para 10

### Métodos Comuns

- `append(item)`: Adiciona um item ao final da lista.
- `insert(index, item)`: Adiciona item no índice especificado.
- `remove(item)`: Remove a primeira ocorrência do item.
- `pop(index)`: Remove e retorna o item no índice especificado.
- `sort()`: Ordena a lista in-place.
- `reverse()`: Inverte a ordem dos elementos da lista.

## List

### Operações Comuns

# Concatenar listas

lista\_concatenada = numeros + palavras

# Repetir listas

lista\_repetida = numeros \* 2

# Tamanho da lista

tamanho = len(numeros)

# Checar se um item está na lista

existe = 3 in numeros

## List

### Fatiamento (Slicing)

- Fatie listas para acessar sublistas.

```
sublista = numeros[1:3] # Retorna [2, 3]
```

### Compreensão de Lista

- Crie listas de forma concisa usando expressões.

```
quadrados = [x**2 for x in numeros] # [1, 4, 9, 16, 25]
```

### Iterando Sobre Listas

- Use loops para percorrer os elementos de uma lista.

```
for numero in numeros:  
    print(numero)
```



## List

### Exemplos Práticos

```
# Criando uma lista de strings  
frutas = ["maçã", "banana", "cereja"]
```

```
# Adicionando uma fruta  
frutas.append("laranja")
```

```
# Removendo uma fruta  
frutas.remove("banana")
```

```
# Ordenando a lista  
frutas.sort()
```

```
print(frutas) # Saída: ['cereja', 'laranja', 'maçã']
```

# Tupla

## O que é uma Tupla?

- Uma **tupla** em Python é uma coleção ordenada e imutável de elementos.
- Diferente das listas, uma vez criada, os elementos de uma tupla não podem ser alterados.
- Tuplas são úteis para armazenar dados que não devem ser modificados.

### Criando uma Tupla

#### # Tupla vazia

```
tupla_vazia = ()
```

# Tupla

## Tupla

**# Tupla com um único elemento (note a vírgula)**

```
tupla_simples = (5,)
```

**# Tupla com múltiplos elementos**

```
tupla_variada = (1, "hello", 3.14)
```

### Acessando Elementos

- Utilize índices para acessar elementos individuais da tupla.
- Índices começam em 0.

```
primeiro = tupla_variada[0] # Retorna 1
```

```
ultimo = tupla_variada[-1] # Retorna 3.14
```

# Tupla

## Tupla

### Imutabilidade

- Tuplas não permitem a alteração de seus elementos após a criação.
- Exemplo: **tupla\_variada[0] = 2** resultará em um erro.

### Métodos Comuns

- **count(valor)**: Conta quantas vezes um valor aparece na tupla.
- **index(valor)**: Retorna o índice da primeira ocorrência do valor.

### Desempacotamento de Tuplas

- Tuplas permitem que múltiplas variáveis sejam atribuídas de uma vez.

```
a, b, c = tupla_variada # a=1, b="hello", c=3.14
```

# Tupla

## Tupla

### Tuplas vs Listas

- **Imutabilidade:** Tuplas são imutáveis, listas são mutáveis.
- **Uso:** Use tuplas quando a integridade dos dados é crítica e você deseja garantir que eles não sejam alterados.

### Operações Comuns

- Concatenar Tuplas: `tupla1 + tupla2`
- Repetir Tuplas: `tupla * n`
- Verificar Existência: `elemento in tupla`

# Tupla

## Tupla

### Exemplo Prático

```
# Tupla de coordenadas
```

```
coordenadas = (10, 20)
```

```
# Desempacotamento de tupla
```

```
x, y = coordenadas
```

```
# Imprimindo valores
```

```
print(f"X: {x}, Y: {y}")
```

# Dict

## O que é um Dicionário?

- Um dicionário em Python é uma coleção de pares **chave:valor**
- É um tipo de dado mutável, o que significa que você pode alterar, adicionar ou remover itens após sua criação.
- Chaves são únicas e imutáveis (strings, números, tuplas), enquanto os valores podem ser de qualquer tipo.

### Criando um Dicionário

```
# Dicionário vazio
```

```
dicionario_vazio = {}
```

```
# Dicionário com elementos
```

```
peessoa = {
```

```
    "nome": "João",
```

```
    "idade": 25,
```

```
    "cidade": "São Paulo"
```

```
}
```

## Dicionário

### Acessando Valores

- Use a chave para acessar o valor correspondente.

```
nome = pessoa["nome"] # Retorna "João"
```

```
pessoa = {  
    "nome": "João",  
    "idade": 25,  
    "cidade": "São Paulo"  
}
```

### Modificando Valores

- Modifique um valor existente ou adicione um novo par **chave:valor**.

```
pessoa["idade"] = 26 # Modifica o valor existente
```

```
pessoa["profissao"] = "Engenheiro" # Adiciona um novo par
```



# Dict

## Dicionário

### Métodos Comuns

- **keys()**: Retorna todas as chaves do dicionário.
- **values()**: Retorna todos os valores do dicionário.
- **items()**: Retorna uma lista de tuplas (chave, valor).
- **get(chave, valor\_default)**: Retorna o valor associado à chave, ou um valor padrão se a chave não existir.
- **pop(chave)**: Remove e retorna o valor associado à chave.

# Dict

## Dicionário

### Iterando Sobre um Dicionário

```
for chave, valor in pessoa.items():  
    print(f"{chave}: {valor}")
```

```
pessoa = {  
    "nome": "João",  
    "idade": 25,  
    "cidade": "São Paulo"  
}
```

### Removendo Elementos

- Use `del` para remover um item específico.

```
del pessoa["cidade"]
```

## Dicionário

### Exemplo Prático

# Criando um dicionário de contatos

```
contatos = {  
    "Ana": "555-1234",  
    "Pedro": "555-5678",  
    "Maria": "555-8765"  
}
```

# Adicionando um novo contato

```
contatos["Lucas"] = "555-4321"
```

# Acessando um número de telefone

```
telefone_ana = contatos.get("Ana")
```

# Removendo um contato

```
contatos.pop("Pedro")
```



## Dicionário

### Exemplo Prático

# Criando um dicionário de contatos

```
contatos = {  
    "Ana": "555-1234",  
    "Pedro": "555-5678",  
    "Maria": "555-8765"  
}
```

# Adicionando um novo contato

```
contatos["Lucas"] = "555-4321"
```

# Acessando um número de telefone

```
telefone_ana = contatos.get("Ana")
```

# Removendo um contato

```
contatos.pop("Pedro")
```



# Set

## O que é um Conjunto (Set)?

- Um **set** em Python é uma coleção desordenada de elementos únicos.
- Conjuntos são mutáveis, o que significa que você pode adicionar ou remover elementos após a criação.
- Eles são usados para armazenar múltiplos itens em uma única variável, semelhante a listas ou dicionários, mas sem duplicatas.

### Criando um Conjunto

```
# Conjunto vazio
```

```
conjunto_vazio = set()
```

```
# Conjunto vazio
```

```
conjunto_vazio = set()
```

```
# Conjunto com elementos
```

```
frutas = {"maçã", "banana", "cereja"}
```

# Set

## Conjunto (Set)

### Características Principais

- **Elementos únicos:** Duplicatas não são permitidas.
- **Desordenado:** A ordem dos elementos não é garantida.
- **Imutabilidade dos elementos:** Os elementos do conjunto devem ser de tipos imutáveis, como strings, números, ou tuplas.

# Set

## Conjunto (Set)

### Métodos Comuns de Sets

- `add(item)`: Adiciona um item ao conjunto.

```
frutas.add("laranja")
```

- `remove(item)`: Remove um item do conjunto (gera um erro se o item não existir).

```
frutas.remove("banana")
```

- `discard(item)`: Remove um item do conjunto (não gera erro se o item não existir).

```
frutas.discard("banana")
```

## Conjunto (Set)

### Métodos Comuns de Sets

- `pop()`: Remove e retorna um item aleatório do conjunto.

```
fruta_removida = frutas.pop()
```

- `clear()`: Remove todos os itens do conjunto.


```
frutas.clear()
```

- `union(another_set)`: Retorna a união de dois conjuntos.

```
conjunto1 = {1, 2, 3}
```

```
conjunto2 = {3, 4, 5}
```

```
uniao = conjunto1.union(conjunto2) # {1, 2, 3, 4, 5}
```





## Conjunto (Set)

### Métodos Comuns de Sets

- `intersection(another_set)`: Retorna a interseção de dois conjuntos.

```
intersecao = conjunto1.intersection(conjunto2) # {3}
```

- `difference(another_set)`: Retorna a diferença entre dois conjuntos.

```
diferenca = conjunto1.difference(conjunto2) # {1, 2}
```

- `issubset(another_set)`: Verifica se um conjunto é subconjunto de outro.

```
subconjunto = {1, 2}
```

```
resultado = subconjunto.issubset(conjunto1) # True
```

## Conjunto (Set)

### Métodos Comuns de Sets

- `issuperset(another_set)`: Verifica se um conjunto é superconjunto de outro.

```
resultado = conjunto1.issuperset(subconjunto) # True
```

- `symmetric_difference(another_set)`: Retorna a diferença simétrica entre dois conjuntos.

```
diff_simetrica = conjunto1.symmetric_difference(conjunto2) # {1, 2, 4, 5}
```



## Conjunto (Set)

### Operações Matemáticas com Conjuntos

- Conjuntos são úteis para operações matemáticas como **união**, **interseção**, **diferença** e **diferença simétrica**.
- Essas operações permitem manipular facilmente coleções de dados.

### Exemplo Prático

```
A = {1, 2, 3, 4}
```

```
B = {3, 4, 5, 6}
```

```
# União
```

```
print(A | B) # {1, 2, 3, 4, 5, 6}
```

```
# Interseção
```

```
print(A & B) # {3, 4}
```

```
# Diferença
```

```
print(A - B) # {1, 2}
```

```
# Diferença Simétrica
```

```
print(A ^ B) # {1, 2, 5, 6}
```

FIAP

THE WAY WE ARE