Conjuntos

Um conjunto é uma coleção **não ordenada** de elementos, **sem elementos repetidos**.

```
conjunto1 = {'a','b','c'}

conjunto2 = set([3, 4, 5, 6])
```

Chaves ({}) podem ser usadas para criar conjuntos diretamente, **quando contiverem elementos**. Use uma lista de elementos separados por vírgulas entre chaves.

A função ***set()*** pode ser usada para criar conjuntos, especialmente **quando precisamos inicializar um conjunto vazio.**

Conjunto vazio

Um conjunto vazio é uma coleção que **não contém elementos**.

```
tipo = type({})
print(tipo) # Saída: <class 'dict'>


conjunto_vazio = set()
```

Chaves {} vazias criam outra estrutura, não um conjunto.

Para um conjunto vazio, use set() **sem argumentos**.

Acessando elementos

Para verificarmos a existência de um item, utilizamos operadores ***in*** e ***not in***.

Considere `conjunto = {1,2,3,4,5}`

Método	Conceito	Exemplo	Saída
in	Verifica se um elemento **está presente** em um conjunto. Ele retorna True se o elemento estiver no conjunto e False caso contrário.	print(3 in conjunto) print(6 in conjunto)	True False
not in	Verifica se um elemento **não está presente em um conjunto**. Ele retorna True se o elemento não estiver no conjunto e False caso contrário.	print(3 not in conjunto) print(6 not in conjunto)	False True

Métodos de manipulação

Conjuntos possuem diversos métodos para adicionar e remover elementos.

Considere `conjunto = {1,2,3}`

Método	Conceito	Exemplo	Conjunto Atual
add(elemento)	Adiciona um único elemento.	conjunto.add(4)	{1, 2, 3, 4}
update(iterável)	Adiciona vários elementos de uma vez.	conjunto.update([5, 6, 7])	{1, 2, 3, 4, 5, 6, 7}
remove(elemento)	Remove um item. Se o elemento não existir, gera um KeyError.	conjunto.remove(2)	{1, 3, 4, 5, 6, 7}
discard(elemento)	Remove um item. Se o elemento não existir, não gera erro.	conjunto.discard(2)	{1, 3, 4, 5, 6, 7}
pop()	Remove e retorna um elemento **aleatório**.	conjunto.pop()	{3, 4, 5, 6, 7}
clear()	Remove todos os elementos do conjunto.	conjunto.clear()	set()

Métodos de manipulação

Existem diversos métodos de operações entre conjuntos.

Considere `a = {1,2,3}` e `b = {3,4,5}`

Método	Conceito	Exemplo	Saída
union(set2) ou (|)	União de dois conjuntos.	a.union(b) a | b	{1, 2, 3, 4, 5}
intersection(set2) ou (&)	Interseção de dois conjuntos.	a.intersection(b) a & b	{3}
difference(set2) ou (-)	Diferença entre dois conjuntos, considerando elementos exclusivos do primeiro conjunto.	a.difference(b) a - b	{1, 2}
symmetric_difference(set2) ou (^)	Diferença entre dois conjuntos, considerando elementos exclusivos de ambos conjuntos.	a.symmetric_difference(b) a ^ b	{1, 2, 4, 5}

Métodos de manipulação

Podemos analisar a relação entre conjuntos e suas características utilizando alguns métodos e criar cópias de conjuntos.

Considere `a = {1,2}` e `b = {1,2,3,4}`

Método	Conceito	Exemplo	Saída
issubset(set2) ou (<=)	Verifica se um conjunto é **subconjunto** de outro.	a.issubset(b) a <= b	True
issuperset(set2) ou (>=)	Verifica se é um conjunto **superconjunto** de outro.	b.issuperset(a) a >= b	True
isdisjoint(set2)	Verifica se os conjuntos **não têm elementos em comum**.	c = {5, 6, 7} a.isdisjoint(c)	True
len(conjunto)	Verifica o número de elementos de um conjunto.	len(a)	2
copy()	Retorna uma **cópia** do conjunto.	copia = a.copy() print(copia)	{1, 2}

Dicionários

Dicionários são estruturas de dados que armazenam **pares chave-valor**, permitindo acesso rápido aos valores por meio das chaves. Eles servem para organizar dados de forma **estruturada**.

Usamos chaves ({}) para **criar dicionários diretamente**, inserindo pares chave-valor separados por vírgulas. Cada par deve ter a chave e o valor separados por dois pontos (:).

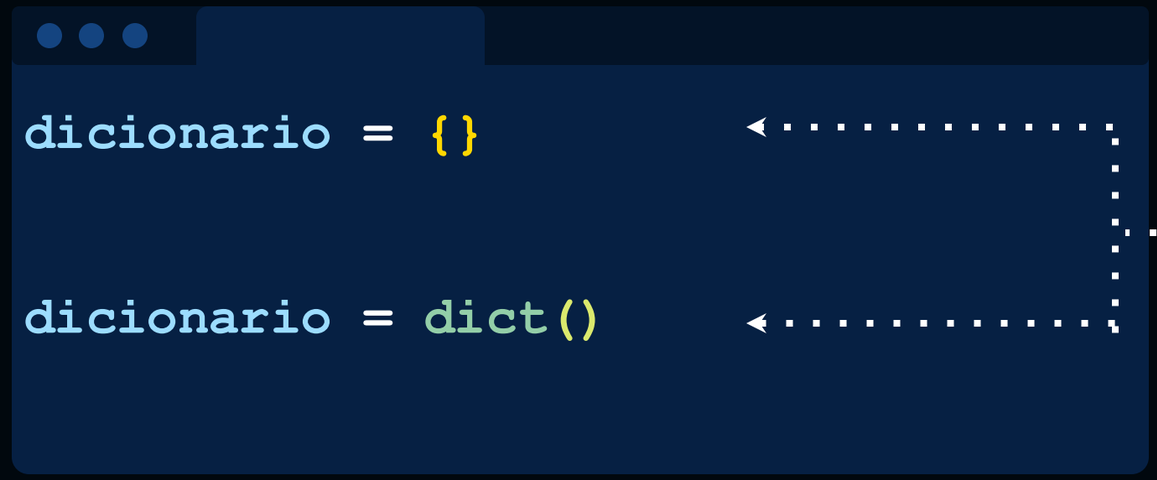
A função ***dict()*** pode ser usada para criar dicionários, passando pares chave-valor como argumentos nomeados ou utilizando uma lista de tuplas.

```python
dic1 = {"nome": "Ana", "nota": 10.0}

dic2 = dict(nome="Eva", nota=9.4)
```

Dicionários vazios

Um dicionário vazio é um dicionário sem nenhum elemento.



Um dicionário vazio pode ser criado usando {} ou dict(). Ele não contém nenhum par chave-valor inicialmente, mas pode ser preenchido conforme necessário.

Criando dicionários com lista de tuplas

Um dicionário pode ser criado a partir de uma lista de tuplas usando ***dict()***. Cada tupla deve conter dois elementos: a chave e o valor.

```python
lista_de_tuplas = [("nome", "Ana"), ("nota", 10.0)]

dicionario = dict(lista_de_tuplas)

print(dicionario) # Saída: {'nome': 'Ana', 'nota': 10.0}
```

Tipos permitidos para chaves

As **chaves** de um dicionário devem ser **imutáveis**. Isso significa que podem ser:

✅ Strings → "nome"
✅ Números (inteiros ou ponto flutuante) → 1 ou 3.14
✅ Tuplas (desde que todos os elementos também sejam imutáveis) → (1, 2, 3)

Mas não podem ser:

❌ Listas e dicionários, por serem **mutáveis**.

```
dicionario_valido = {1: "um", "dois": 2, (3, 4): "par"}

dicionario_invalido = {[1, 2]: "lista"}
# Saída: TypeError: unhashable type: 'list'
```

Tipos permitidos para valores

Os valores de um dicionário podem ser **qualquer tipo de dado**, incluindo:

- ✅ Strings → "nome"
- ✅ Números (inteiros ou ponto flutuante) → 1 ou 3.14
- ✅ Tuplas → (1, 2, 3)
- ✅ Listas e dicionários → [1, 2, 3] ou {"chave": "valor"}
- ✅ Objetos personalizados → Pessoa("Alice")

```
dicionario = {
    "nome": "João",
    "idade": 25,
    "notas": [10.0, 6.3, 7.5],
    "endereco": {"cidade": "São Paulo", "bairro": "Centro"}
}
```

Acessando e modificando dados

Podemos acessar diretamente o valor associado a uma chave usando colchetes [], seguindo o padrão **dicionario[chave] = valor**.

Considere `dic = {"nome": "Ana", "nota": 10.0}`

Método	Conceito	Exemplo	Saída
nome_dicionario[chave]	Acessar um valor por uma chave.	dic["nome"]	Ana
nome_dicionario[chave] = valor	Atribuir um valor de uma chave.	dic["nota"] = 9.2	{"nome" : "Ana", "nota" : 9.2}
del nome_dicionario[chave]	Remover um Item Específico	del dic["nome"]	{"nota" : 9.2}

Acessando e modificando dados

Considere `dic = {"nome": "Ana", "nota": 10.0}`

Método	Conceito	Exemplo	Saída
in	Verifica se uma chave **está presente** no dicionário. Ele retorna True se a chave estiver e False caso contrário.	print("nome" in dic) print("idade" in dic)	True False
not in	Verifica se uma chave **não está presente** em um dicionário. Ele retorna True se a chave não estiver e False caso contrário.	print("idade" not in dic)	True

Métodos de manipulação

Dicionários possuem diversos métodos para acessar seus dados e verificar seu tamanho.

Considere `dicionario = {"nome": "João", "idade": 28}`

Método	Conceito	Exemplo	Saída
keys()	Retorna as chaves do dicionário.	dicionario.keys()	*dict_keys(['nome', 'idade'])*
values()	Retorna os valores do dicionário.	dicionario.values()	*dict_values(['João', 28])*
items()	Retorna pares (chave, valor) como tuplas.	dicionario.items()	*dict_items([('nome', 'João'), ('idade', 28)])*
len(dicionario)	Contar quantos pares (chave, valor) o dicionário possui.	len(dicionario)	2

Métodos de manipulação

Além de acessar valores, podemos buscar e modificar elementos.

Considere `dicionario = {"nome": "João", "idade": 28}`

Método	Conceito	Exemplo	Saída
get(chave, padrão=None)	Obtém um valor sem erro se a chave não existir.	dicionario.get("nome") dicionario.get("altura", "Não Informado!")	Ana Não Informado!
pop(chave, padrão)	Remove uma chave e retorna seu valor. Se o padrão não for fornecido e a chave não estiver no dicionário, um KeyError é levantado.	dicionario.pop("idade")	{"nome" : "João"}
list(dicionario)	Usada para obter todas as chaves do dicionário em forma de lista.	chaves = list(dicionario) print(chaves)	["nome"]
clear()	Apaga todos os itens do dicionário.	dicionario.clear()	{}

Métodos de manipulação

Dicionários também permitem atualizar e remover elementos de diferentes maneiras.

Considere `dicionario = {"a": 1, "b": 2, "c": 3}`

Método	Conceito	Exemplo	Saída
popitem()	Remove e retorna o último par (chave, valor) inserido. Se o dicionário estiver vazio, um KeyError é levantado.	dicionario.popitem()	{"a": 1, "b": 2}
reversed(dicionario)	Retorna um iterador sobre as chaves do dicionário em ordem reversa.	print(list(reversed(dicionario)))	["b", "a"]
update([outro])	Atualiza o dicionário com outro mapeamento.	dicionario.update({"a": 1.0, "d": 4})	{"a": 1.0, "b": 2, "d": 4}
copy()	Retorna uma cópia rasa do dicionário.	copia = dicionario.copy() print(copia)	{"a": 1.0, "b": 2, "d": 4}

Criando um estoque com conjuntos e dicionários

```
estoque = {
    "frutas": {"maça", "uva"},
    "verduras": {"cenoura", "alface"}
} # Dicionário com conjuntos

estoque["frutas"].add("morango") # Adicionando um item

estoque["verduras"].discard("alface")# Removendo um item

print(estoque)
```

Saída: {'frutas': {'maça', 'morango', 'uva'}, 'verduras': {'cenoura'}}

alura

Compartilhe um resumo de seus novos conhecimentos em suas redes sociais.

#aprendizadoalura