|  |  |
| --- | --- |
|  | **Dicionário** |
| Mutável | Sim |
| Definir | Dicionario = {“Chave”: “Valor”} | Aluno={“Nome”: “Joaquim”, “Idade”:15} |
| Mostrar um | Dicionario[“Chave”] | Aluno[“Nome”] | (Dá erro se não existir )  Aluno.get(“Morada”, “Não tem morada definida”) | (Não dá erro caso a chave não exista) |
| Adicionar | Dicionario[“Nova Chave”] = “Novo Valor” | Aluno[“Morada”] = “Viseu” |
| Remover | del Dicionario[“Nova Chave”] | del Aluno[“Morada”] | (Dá erro se não existir)  Valor = Dicionario.pop(“Chave”, None) | Morada = Aluno.pop(“Morada”, None)  (Se a chave não existir a variável fica na morada com None) |
| Editar | Dicionario[“Chave”] = “Novo Valor”  Aluno[“Idade”] = 16 |
| Listar | for Chave, Valor in Dicionario.items():  print(f”{Chave}:{Valor}”)  for Chave in Dicionario:  print(f”{Chave}:{Dicionario[Chave]}”) |
| Testar se existe | If “Morada” in Aluno:  print(“O aluno tem o campo Morada”)  Valores = Aluno.values()  If “Viseu” in Valores:  Print(“O dicionário tem o valor Viseu”) |
| Igualdade | Operador ==  A ordem dos elementos não interessa para eles serem iguais |
| Funções | Dict() – Cria / Converte um dicionario  Keys() – Lista das chaves do dicionario  Values() - Lista dos valores do dicionario  Update() – Junta dois dicionarios |
| Notas | Juntar dois dicionários com o operador |  A chave tem de ser um valor imutavel |

|  |  |
| --- | --- |
|  | **Conjunto (Set)** |
| Mutável | Sim |
| Definir | Conjunto = {1,2,3,4,5} | Nomes = {“Joaquim”, “Maria”, “Antonio”} |
| Mostrar um | Não é possivel indexer  Mostrar o 3º elemento  For Pos, Valor in enumerate(Conjunto, 0):  If Pos == 2:  Print(Valor)  Break |
| Adicionar | Conjunto.add(10)  Nomes.add(“Mais Um”) |
| Remover | Conjunto.remove(3) – Remover o valor “3”  Nomes.remove(“Joaquim”) – Remove o nome “Joaquim” do conjunto |
| Editar | Primeiro remover o elemento e depois criar um novo |
| Listar | for Pos, Valor in enumerate(Nomes, 0)  print(f”{Pos} : {Valor}”) |
| Testar se existe | Operador in  If “Joaquim” in Nomes:  Print(“O conjunto tem um Joaquim”) |
| Igualdade | Operador ==  A ordem dos elementos não interessa |
| Funções | União: resultado = Conjunto.union(Outro\_Conjunto)  Interseção: resultado = Conjunto.intersection(Outro\_Conjunto)  Diferença: resultado = Conjunto.diference(Outro\_Conjunto)  Diferença simétrica: resultado = Conjunto.symmetric\_difference(Outro\_Conjunto) |
| Notas | Coleção não ordenada, que não garante as posições dos elementos, todos os elementos  têm de ser imutáveis e repitidos |

|  |  |
| --- | --- |
|  | **Tuplo** |
| Mutável | Não |
| Definir | Meu\_Tuple = 1,2,3  Outro\_Tuple = (1,2,3,4) |
| Mostrar um | Print(Meu\_Tuple[0]) |
| Adicionar | Não se aplica |
| Remover | Não se aplica |
| Editar | Não se aplica |
| Listar | For Valor in Meu\_Tuple:  Print(Valor)  For i in range(len(Meu\_Tuple)):  Print(Meu\_Tuple[i] |
| Testar se existe | Operador in  If 2 in Meu\_Tuple  Print(“Meu Tuple tem um 2”) |
| Igualdade | Operador ==  A ordem dos elementos não interessa |
| Funções | Max – Devolve o Maior | max(Meu\_Tuple)  Min – Devolve o Menor | min(Meu\_Tuple)  Count – Contar o Nº de ocorrencias | Meu\_Tuple.count(2)  Sum – Devolve a soma de todos os valores | sum(Meu\_Tuple)  Operador + - Para juntar dois tuples | Resultado = Meu\_Tuple + Outro\_Tuple |
| Notas | São imutáveis, ou seja, depois de criado não pode ser alterado  Pode ter valores repitidos |

|  |  |
| --- | --- |
|  | **Lista** |
| Mutável | Sim |
| Definir | Minha\_Lista = [1,2,3,4,5,6,7,8] |
| Mostrar um | Print(Minha\_Lista[2]) |
| Adicionar | Minha\_Lista.append(10) |
| Remover | Minha\_Lista.remove(10) |
| Editar | Minha\_Lista[2] = 10 |
| Listar | For Valor in Minha\_Lista:  Print(Valor)  For i in range(len(Minha\_Lista)):  Print(Minha\_Lista[i]) |
| Testar se existe | Operador in  If 2 in Minha\_Lista:  Print(“Existe o 2”) |
| Igualdade | Operador ==  A ordem dos elementos não interessa |
| Funções | Max – Devolve o Maior | max(Meu\_Tuple)  Min – Devolve o Menor | min(Meu\_Tuple)  Count – Contar o Nº de ocorrencias | Meu\_Tuple.count(2)  Sum – Devolve a soma de todos os valores | sum(Meu\_Tuple)  Operador + - Para juntar dois tuples | Resultado = Meu\_Tuple + Outro\_Tuple  Slicing |
| Notas | Podem ter elementos mutáveis (Dicionarios, Tuples, Listas, Arrays) |