Listas, Dicionários, Tuplas e Conjuntos

Tuplas

Semelhantes as listas, porém são **imutáveis**: não se pode acrescentar, apagar ou fazer atribuições aos itens. Tuplas são ideais para representar listas de valores constantes. Suportam a maior parte das operações de lista, como fatiamento e indexação.

> 💡 **Mas tuplas não podem ter seus elementos alterados.**

```
tupla = (a,b,...,z)

dias=('domingo', 'segunda','terça','quarta','quinta','sexta',
dias='domingo', 'segunda','terça','quarta','quinta','sexta','
```

```
#Particularidade: tupla com apenas um elemento
t1 = (1,)

#tuplas vazias
t=()

#Os elementos são referenciado da mesma forma que os elemento
primeiro_elemento= tupla[0]

#Lista podem ser convertidas em tuplas
tupla = tuple(lista)

#Tuplas podem ser convertidas em listas
lista = list(tupla)
```

As **tuplas** são mais eficientes do que as listas convencionais, pois consomem menos recursos computacionais (memória), por serem estruturas mais simples, tal como as *strings* imutáveis em relação às *strings* mutáveis.

```
#É possível criar tuplas que contenham objetos mutáveis
lista=[3,4]
tupla=(1,2,lista)
lista.extend([5,6])
print(tupla)
tupla[2]=[3,4] #erro de compilação
```

Tuplas também podem ser utilizadas para desempacotar valores, por exemplo:

```
a,b = 10,20
a
b
a,b=b,a
a,b=[10,20]
*a,b=[1,2,3,4,5]
a,*b=[1,2,3,4,5]
#*a, b, dizemos: coloque o último valor em b e os restantes e
#a, *b, dizemos: coloque o primeiro valor em a e os outros em
a,*b,c=[1,2,3,4,5]
```

Range

O range é um tipo de sequência imutável de números, sendo comumente usado para *looping* de um número específico de vezes em um comando for já que representam um intervalo. O comando **range** gera um valor contendo números inteiros sequenciais, obedecendo a sintaxe:

```
##range(início, fim,passos)

for valor in range(1,10,2):
    print(valor)
```

Conjuntos

O Python provê entre os *builtins* também: `set` **(Conjuntos).** Sequência mutável unívoca (sem repetições) não ordenada. Os dois tipos implementam operações de conjuntos, tais como: união, interseção e diferença.

```
frutas = {'laranja', 'banana','uva','pera','laranja'}
frutas.add('manga')
a = set('abacate')
b = set('abacaxi')
#diferença
```

```
a-b
#uniao
a|b
#interseção
a&b
#diferença simétrica
a^b
```

```
#conjunto de dados
s1 = set(range(3))
s2 = set(range(10,7,-1))
s3 = set(range(2,10,2))

#exibe os dados
print('s1:', s1, '\ns2: ', s2, '\ns3: ',s3)

#união
s1s2 = s1.union(s2)
print('União de s1 e s2: ', s1s2)

#diferença
print('Diferença com s3: ', s1s2.difference(s3))

#intersecção
print('Intersecção ocm s3: ', s1s2.intersection(s3))

#testa se um set inclui outro
if s1.issuperset([1,2]):
    print('s1 inclui 1 e 2')

#testa se não existe elementos em comum
if s1.isdisjoint(s2):
    print('s1 e s2 não tem elementos em comum')
```

Dicionários

Um **dicionário** é uma lista de associações compostas por uma chave única e estruturas correspondentes. Dicionários são mutáveis, tais como as listas.

A chave precisa ser de um tipo imutável, geralmente são usadas ***strings***, mas também podem ser **tuplas** ou tipos numéricos. Os dicionários pertencem ao tipo de mapeamento integrado e não sequenciais como as **listas**, **tuplas** e ***strings***.

Os dicionários são delimitados por chaves (`{}`) e suas chaves ('nome', 'idade' e 'cidade') por aspas. Já os valores podem ser de qualquer tipo.

```
dicionario = {'a':a, 'b':b, ..., 'z:',z}

pessoa={'nome':'João','idade':25,'cidade': 'São Paulo'}
pessoa['Nome']
pessoa['idade']
pessoa['país']='Brasil'

a=dict(um=1,dois=2,três=3)
```

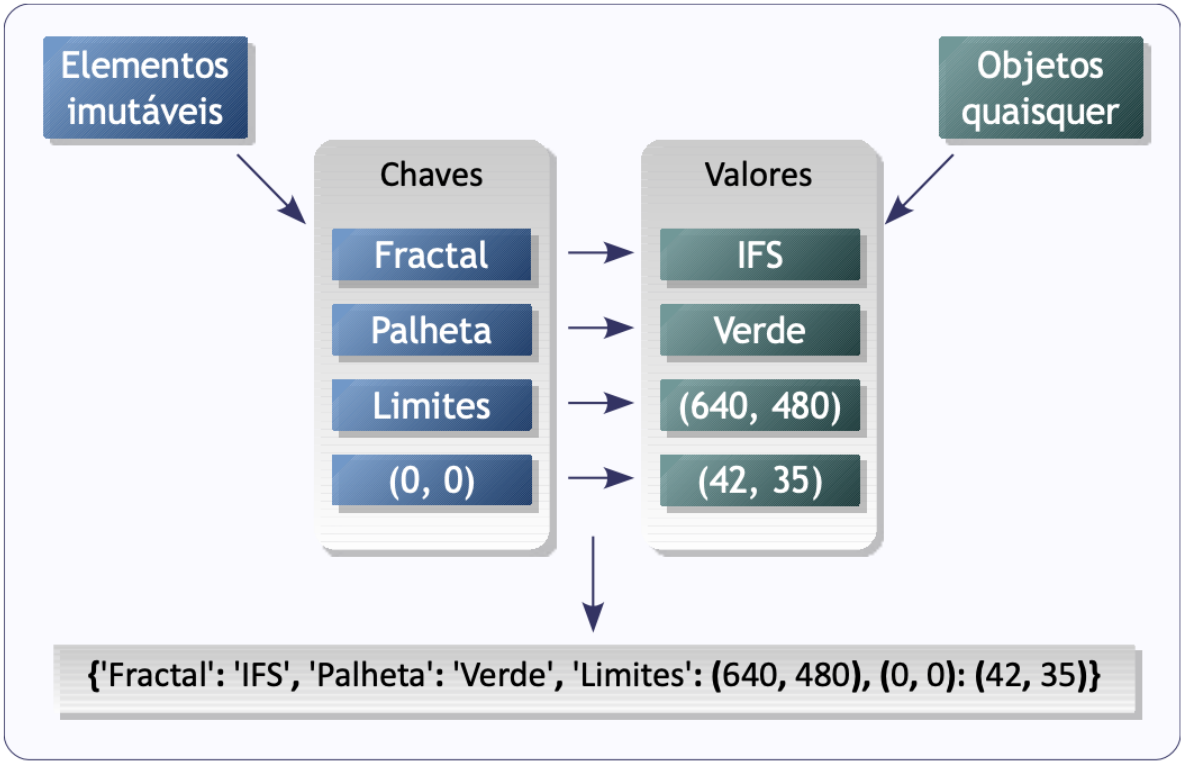
```
tabela = {"Alface": 0.45,
                    "Batata": 1.20,
                    "Tomate": 2.30,
                    "Feijão": 1.50}

print(tabela["Tomate"])

tabela["Tomate"]=2.50
print(tabela["Tomate"])

tabela["Cebola"]=2.50
print(tabela)
```

```
print("Manga" in tabela)
print("Batata" in tabela)
```




```
#exemplo de dicionário
dic = {'nome': 'Billie Joe', 'Banda':'Green Day'}

#acessando elementos
print(dic['nome'])

#adicionando elementos
dic['album'] = 'American Idiot'

#apagando um elemento do dicionário
del dic['album']

#tipos
```

```
itens = dic.items()
chaves = dic.keys()
valores=dic.values()
```

```
#bandas e seus albuns
bandas={'Green Day':['Dookie', 'American Idiot'],
                'AC/DC': ['Black Ice','Rock or Burst'],
                'Pink Floyd':['The Wall','The Dark Side of th

#mais bandas
bandas['Foo Fighters'] = ['One by One','In Your Honor']

#items() retorna uma lista de tuplas com a chave e o valor
for banda, albuns in bandas.items()
    print(banda, '=>', albuns)

#se tiver AC/DC deleta
if bandas.has_key('AC/DC'):
    del bandas['AC/DC']
```

Dicionários com Listas

Em Python, podemos ter dicionários nos quais as chaves são associadas a listas ou mesmo a outros dicionários.

```
#a chave é o nome do produto e a lista contém a quantidade e
estoque={"tomate": [1000,2.30],
                "alface": [500,0.45],
                "batata": [2001,1.20],
                "feijão": [100, 1.50]}

venda = [["tomate",5],["batata",10],["alface",5]]
total=0
```

```
print("Vendas:\n")
for operacao in venda:
    produto, quantidade = operacao
    preco = estoque[produto][1]
    custo = preco*quantidade
    print(f"{produto:12s}:{quantidade:3d}x{preco:6.2f}={custo
    estoque[produto][0]-=quantidade
    total +=custo
print(f"Custo total: {total:21.2f}\n")
print("Estoque:\n")
for chave, dados in estoque.items():
    print("Descrição: ", chave)
    print("Quantidade: ", dados[0])
    print(f"Preço: {dados[1]:6.2f}\n")
```

```
#Exemplo de dicionário sem valor padrão
d={}
for letra in "abracadabra":
    if letra in d:
        d[letra] = d[letra] +1
    else:
        d[letra] =1

print(d)

#outra forma
d={}
for letra in "abracadabra"
    d[letra] = d.get(letra,0)+1
print(d)
```

O método `get` tenta obter a chave procurada; caso não a encontre, retorna o segundo parâmetro, no caso 0 (zero). Se o segundo parâmetro não for especificado, `get` retornará **None.**

Observações

Em Python, o tipo booleano (*bool*) é uma especialização do tipo inteiro (*int*). O verdadeiro é chamado `True` e é igual a `1`, enquanto o falso é chamado `False` e é igual a zero.

Os seguintes valores são considerados falsos:

- *False* (falso).
- *None* (nulo).
- 0 (zero).
- '' (*string* vazia).
- [] (lista vazia).
- () (tupla vazia).
- {} (dicionário vazio).

Outras estruturas com o tamanho igual a zero.

Além dos operadores booleanos, existem as funções `all()`, que retorna verdadeiro quando todos os itens forem verdadeiros na sequência usada como parâmetro, e `any()`, que retorna verdadeiro se algum item o for.

Resumo das estruturas de dados

	Listas	Tuplas	Dicionários	Conjuntos
Ordem dos elementos	Fixa	Fixa	Mantida a partir do Python 3.7	Indeterminada
Tamanho	Variável	Fixo	Variável	Variável
Elementos repetidos	Sim	Sim	Pode repetir valores, mas as chaves devem ser únicas	Não
Pesquisa	Sequencial, índice numérico	Sequencial, índice numérico	Direta por chave	Direta por valor
Alterações	Sim	Não	Sim	Sim
Uso primário	Sequências	Sequências constantes	Dados indexados por chave	Verificação de unicidade, operações com conjuntos