Criando uma Tupla

Podemos criar uma tupla usando as duas maneiras

1. **Usando parênteses ():** Uma tupla é criada colocando itens separados por vírgulas entre colchetes arredondados.
2. Usando um `tuple()`construtor: crie uma tupla passando os itens separados por vírgula dentro do arquivo `tuple()`.

Exemplo

Uma tupla pode ter itens de diferentes tipos de dados integer, float, list, string, etc;

```
# create a tuple using ()
# number tuple
number_tuple = (10, 20, 25.75)
print(number_tuple)
# Output (10, 20, 25.75)

# string tuple
string_tuple = ('Jessa', 'Emma', 'Kelly')
print(string_tuple)
# Output ('Jessa', 'Emma', 'Kelly')

# mixed type tuple
sample_tuple = ('Jessa', 30, 45.75, [25, 78])
print(sample_tuple)
# Output ('Jessa', 30, 45.75, [25, 78])

# create a tuple using tuple() constructor
sample_tuple2 = tuple(('Jessa', 30, 45.75, [23, 78]))
print(sample_tuple2)
# Output ('Jessa', 30, 45.75, [23, 78])
```

Como podemos ver na saída acima, os diferentes itens são adicionados na tupla como integer, string e list.

Criar uma tupla com um único item

Uma tupla de item único é criada colocando um item entre parênteses seguido por uma vírgula. Se o tempo da tupla for uma string entre parênteses

e não seguida por uma vírgula, o Python o tratará como um `str`tipo. Vejamos isso com um exemplo.

```
# without comma
single_tuple = ('Hello')
print(type(single_tuple))
# Output class 'str'
print(single_tuple)
# Output Hello

# with comma
single_tuple1 = ('Hello',)
# output class 'tuple'
print(type(single_tuple1))
# Output ('Hello',)
print(single_tuple1)
```

Como podemos ver na saída acima pela primeira vez, não adicionamos uma vírgula após o “Hello”. Assim, o tipo de variável era class `str`, e na segunda vez era uma classe `tuple`.

Embalar e desembalar

Uma tupla também pode ser criada sem usar um `tuple()`construtor ou colocar os itens entre parênteses. Ela é chamada de variável “Embalagem”.

Em Python, podemos criar uma tupla empacotando um grupo de variáveis. O empacotamento pode ser usado quando queremos coletar vários valores em uma única variável. Geralmente, essa operação é chamada de empacotamento de tuplas.

Da mesma forma, podemos descompactar os itens apenas atribuindo os itens da tupla ao mesmo número de variáveis. Esse processo é chamado de “Descompactação”.

Vejamos isso com um exemplo.

```
# packing variables into tuple
tuple1 = 1, 2, "Hello"
# display tuple
print(tuple1)
# Output (1, 2, 'Hello')

print(type(tuple1))
```

```
# Output class 'tuple'

# unpacking tuple into variable
i, j, k = tuple1
# printing the variables
print(i, j, k)
# Output 1 2 Hello
```

Como podemos ver na saída acima, três itens de tupla são atribuídos a variáveis individuais i, j, k, respectivamente.

Caso atribuamos menos variáveis do que o número de itens na tupla, obteremos o erro de valor com a mensagem muitos valores para descompactar

Comprimento de uma Tupla

Podemos encontrar o comprimento da tupla usando a `len()`função. Isso retornará o número de itens na tupla.

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
# length of a tuple
print(len(tuple1))
# Output 6
```

Iterando uma Tupla

Podemos iterar uma tupla usando um loop for Vamos ver isso com um exemplo.

```
# create a tuple
sample_tuple = tuple((1, 2, 3, "Hello", [4, 8, 16]))
# iterate a tuple
for item in sample_tuple:
    print(item)
```

Saída

```
1

2

3

Olá

[4, 8, 16]
```

Como podemos ver na saída acima, estamos imprimindo cada item da tupla usando um loop.

Acessando itens de uma Tupla

A tupla pode ser acessada por meio de indexação e fatiamento. Esta seção irá guiá-lo acessando a tupla usando as duas maneiras a seguir

- **Usando indexação** , podemos acessar qualquer item de uma tupla usando seu número de índice
- **Usando slicing** , podemos acessar uma variedade de itens de uma tupla

Indexação

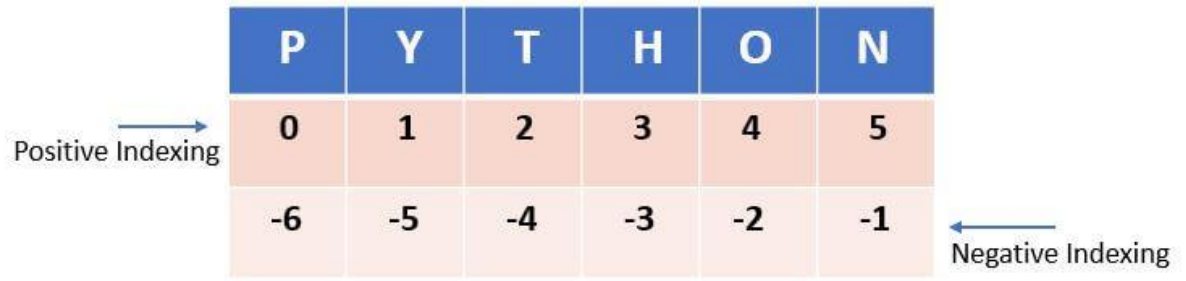
Uma tupla é uma sequência ordenada de itens, o que significa que eles mantêm a ordem de inserção dos dados. Ele mantém o valor de índice para cada item.

Podemos acessar um item de uma tupla usando seu número de índice dentro do operador de índice `[]`e esse processo é chamado de **“Indexação”** .

Nota :

- Como as tuplas são sequências ordenadas de itens, os valores de índice começam de 0 até o comprimento da tupla.
- Sempre que tentamos acessar um item com um índice maior que o comprimento da tupla, ele lançará o arquivo `'Index Error'`.

- Da mesma forma, os valores de índice são sempre inteiros. Se dermos qualquer outro tipo, ele lançará `Type Error`.



Na imagem acima, podemos ver que os valores do índice começam do zero e vão até o último item cujo valor do índice será `len(tuple) - 1` .

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
for i in range(4):
    print(tuple1[i])
```

Saída

```
P

S

T

H
```

Como visto no exemplo acima, imprimimos os primeiros quatro itens da tupla com a indexação.

Nota : Se mencionarmos o valor do índice maior que o comprimento de uma tupla, ele lançará um erro de índice.

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')

# IndexError: tuple index out of range
print(tuple1[7])
```

Além disso, se você mencionar qualquer valor de índice diferente de inteiro, ele lançará um erro de tipo.

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')

# TypeError: tuple indices must be integers or slices, not float
print(tuple1[2.0])
```

Indexação negativa

Os valores de índice também podem ser negativos, com o último mas o primeiro item tendo o valor de índice como -1 e penúltimo -2 e assim por diante.

Por exemplo, podemos acessar o último item de uma tupla usando .`tuple_name[-1]`

Vamos fazer duas coisas aqui

- Acesse itens de tupla usando o valor de índice negativo
- Iterar tupla usando indexação negativa

Exemplo

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
# Negative indexing
# print last item of a tuple
print(tuple1[-1])  # N
# print second last
print(tuple1[-2])  # O

# iterate a tuple using negative indexing
for i in range(-6, 0):
    print(tuple1[i], end=", ")
# Output P, Y, T, H, O, N,
```

Fatiar uma tupla

Podemos até especificar um intervalo de itens a serem acessados de uma tupla usando a técnica chamada 'Slicing'. O operador utilizado é `':'`.

Podemos especificar os valores inicial e final para o **intervalo de itens a serem acessados da tupla** . A saída será uma tupla e incluirá o intervalo de itens com os valores de índice desde o início até o final do intervalo. O item de valor final será excluído.

Devemos ter em mente que o valor do índice sempre começa com 0.

Para facilitar o entendimento, usaremos uma tupla inteira com valores de 0 a 9 semelhantes a como um valor de índice é atribuído.

```
tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple with start and end index number
print(tuple1[1:5])
# Output (1, 2, 3, 4)
```

Como visto na saída acima, os valores começando de 1 a 4 são impressos. Aqui o último valor no intervalo 5 é excluído.

Nota :

- Se o valor inicial não for mencionado ao fatiar uma tupla, os valores nas tuplas começarão do primeiro item até o item final no intervalo. Novamente, o item final no intervalo será excluído.
- Da mesma forma, podemos mencionar um intervalo de fatiamento sem o valor final. Nesse caso, será retornado o item com o índice mencionado no valor inicial do intervalo até o final da tupla.

Exemplo

```
tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple without start index
print(tuple1[:5])
# Output (0, 1, 2, 3, 4)

# slice a tuple without end index
print(tuple1[6:])
# Output (6, 7, 8, 9, 10)
```

Da mesma forma, também podemos fatiar a tupla usando indexação negativa. O último mas primeiro item terá o índice -1.

```
tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple using negative indexing
print(tuple1[-5:-1])
# Output (6, 7, 8, 9)
```

Aqui podemos ver que os itens com índices negativos de -1 a -4 são impressos excluindo -5.

Encontrando um item em uma Tupla

Podemos procurar um determinado item em uma tupla usando o `index()`método e ele retornará a posição desse item específico na tupla.

O `index()`método aceita os três argumentos a seguir

1. **item** – O item que precisa ser pesquisado
2. **start** – (Opcional) O valor inicial do índice a partir do qual a pesquisa será iniciada
3. **end** – (Opcional) O valor final da pesquisa de índice

Exemplo

```
tuple1 = (10, 20, 30, 40, 50)

# get index of item 30
position = tuple1.index(30)
print(position)
# Output 2
```

Como visto na saída acima, o valor de índice do item 30 é impresso.

Encontrar dentro de um intervalo

Podemos mencionar os valores inicial e final do `index()`método para que nossa busca seja limitada a esses valores.

Exemplo

```
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
# Limit the search locations using start and end
# search only from location 4 to 6
# start = 4 and end = 6
# get index of item 60
position = tuple1.index(60, 4, 6)
print(position)
# Output 5
```

Como visto na saída acima, limitamos a pesquisa da posição de índice 4 a 6, pois o número 60 está presente apenas neste intervalo. Caso mencionemos algum item que não esteja presente, ele lançará um erro de valor.

```
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
#index out of range
position= tuple1 .index(10)
print(postion)
# Output ValueError: tuple.index(x): x not in tuple
```

Verificando se um item existe

Podemos verificar se um item existe em uma tupla usando o `in`operador. Isso retornará um booleano `True`se o item existir e `False`se não existir.

```
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
# checking whether item 50 exists in tuple
print(50 in tuple1)
# Output True
print(500 in tuple1)
# Output False
```

Como visto na saída acima, podemos ver que o item '50' existe na tupla, portanto, obtivemos `True`e '500' não e, portanto, obtivemos `False`.

Adicionando e alterando itens em uma Tupla

Uma lista é um tipo mutável, o que significa que podemos adicionar ou modificar valores nela, mas as tuplas são imutáveis, portanto, não podem ser alteradas.

Além disso, como uma tupla é imutável, não há métodos internos para adicionar itens à tupla.

Se você tentar modificar o valor, receberá um erro.

Exemplo

```
tuple1 = (0, 1, 2, 3, 4, 5)
tuple1[1] = 10
# Output TypeError: 'tuple' object does not support item assignment
```

Como solução alternativa, podemos converter a tupla em uma lista, adicionar itens e convertê-la novamente em uma tupla. Como as tuplas são coleções ordenadas como listas, os itens sempre são adicionados no final.

```
tuple1 = (0, 1, 2, 3, 4, 5)

# converting tuple into a list
sample_list = list(tuple1)
# add item to list
sample_list.append(6)

# converting list back into a tuple
tuple1 = tuple(sample_list)
print(tuple1)
# Output (0, 1, 2, 3, 4, 5, 6)
```

Como podemos ver na saída acima, o item é adicionado à tupla no final.

Modificar itens aninhados de uma tupla

Uma coisa a ser lembrada aqui, se um dos itens for um tipo de dados mutável como uma lista, podemos alterar seus valores no caso de uma tupla aninhada.

Por exemplo, vamos supor que você tenha a seguinte tupla que tem uma lista como seu último item e você deseja modificar os itens da lista.

```
tupla1 = (10, 20, [25, 75, 85] )
```

Vamos ver como modificar o item de conjunto se ele contiver tipos mutáveis.

Exemplo

```
tuple1 = (10, 20, [25, 75, 85])
# before update
print(tuple1)
# Output (10, 20, [25, 75, 85])

# modify last item's first value
tuple1[2][0] = 250
# after update
print(tuple1)
# Output (10, 20, [250, 75, 85])
```

Como as tuplas são imutáveis, não podemos alterar os valores dos itens na tupla. Novamente com a mesma solução, podemos convertê-lo em uma lista, fazer alterações e convertê-lo novamente em uma tupla.

```
tuple1 = (0, 1, 2, 3, 4, 5)

# converting tuple into a list
sample_list = list(tuple1)
# modify 2nd item
sample_list[1] = 10

# converting list back into a tuple
tuple1 = tuple(sample_list)
print(tuple1)
# Output (0, 10, 2, 3, 4, 5)
```

Como podemos ver na saída acima, o último item foi atualizado de 3 para 11.

Removendo itens de uma tupla

`pop()`Tuplas são imutáveis, portanto, não há `remove()`métodos para a tupla. Podemos remover os itens de uma tupla usando as duas maneiras a seguir.

1. Usando a palavra-chave del
2. Ao convertê-lo em uma lista

Usando a palavra-chave del

A `del`palavra-chave excluirá a tupla inteira.

```
sampletup1 =(0,1,2,3,4,5,6,7,8,9,10)
del sampletup1

print(sampletup1)
```

Saída

```
      3

----> 4 print(amostraup1)



NameError: o nome 'sampletup1' não está definido
```

Como visto na saída acima, estamos recebendo um erro quando tentamos acessar uma tupla excluída.

Ao convertê-lo em uma lista

Podemos converter uma tupla em uma lista e, em seguida, remover qualquer item usando o `remove()`método. Então, novamente, vamos convertê-lo de volta em uma tupla usando o `tuple()`construtor.

```
tuple1 = (0, 1, 2, 3, 4, 5)

# converting tuple into a list
sample_list = list(tuple1)
# reomve 2nd item
sample_list.remove(2)

# converting list back into a tuple
tuple1 = tuple(sample_list)
print(tuple1)
# Output (0, 1, 3, 4, 5)
```

Como visto na saída acima, o item 3 foi removido da tupla.

Contar a ocorrência de um item em uma tupla

Como aprendemos, uma tupla pode conter itens duplicados. Para determinar quantas vezes um item específico ocorreu em uma tupla, podemos usar o `count()`método de um objeto de tupla.

O `count()`método aceita qualquer valor como parâmetro e retorna o número de vezes que um determinado valor aparece em uma tupla.

Exemplo

```
tuple1 = (10, 20, 60, 30, 60, 40, 60)
# Count all occurrences of item 60
count = tuple1.count(60)
print(count)
# Output 3

count = tuple1.count(600)
print(count)
# Output 0
```

Copiando uma tupla

Podemos criar uma cópia de uma tupla usando o operador de atribuição `'='`. Essa operação criará apenas uma cópia de referência e não uma cópia profunda porque as tuplas são imutáveis.

```
tuple1 = (0, 1, 2, 3, 4, 5)

# copy tuple
tuple2 = tuple1
print(tuple2)
# Output (0, 1, 2, 3, 4, 5)

# changing tuple2
# converting it into a list
sample_list = list(tuple2)
sample_list.append(6)

# converting list back into a tuple2
tuple2 = tuple(sample_list)

# printing the two tuples
print(tuple1)
# Output (0, 1, 2, 3, 4, 5)
print(tuple2)
# Output (0, 1, 2, 3, 4, 5, 6)
```

Como podemos ver na saída acima, o tuple1 não é afetado pelas alterações feitas no tuple2.

Concatenar duas Tuplas

Podemos concatenar duas ou mais tuplas de maneiras diferentes. Uma coisa a notar aqui é que as tuplas permitem duplicatas, então se duas tuplas tiverem o mesmo item, ele será repetido duas vezes na tupla resultante. Vejamos cada um deles com um pequeno exemplo.

Usando o `+`operador

Podemos adicionar duas tuplas usando o operador +. Este é um método muito simples e a tupla resultante terá itens de ambas as tuplas.

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)
```

```
# concatenate tuples using + operator
tuple3 = tuple1 + tuple2
print(tuple3)
# Output (1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

Como visto na saída acima, a tupla resultante tem itens de ambas as tuplas e o item 3, 4, 5 é repetido duas vezes.

Usando a função soma()

Também podemos usar a função interna do Python `sum`para concatenar duas tuplas. Mas a função soma de dois iteráveis como tuplas sempre precisa começar com Tupla Vazia. Vejamos isso com um exemplo.

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)

# using sum function
tuple3 = sum((tuple1, tuple2), ())
print(tuple3)
# Output (1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

Como podemos ver na saída acima, a função sum recebe uma tupla vazia como argumento e retorna os itens de ambas as tuplas.

Usando a função chain()

A `chain()`função faz parte do módulo itertools em python. Ele faz um iterador, que retornará todos os primeiros itens iteráveis (uma tupla no nosso caso), que será seguido pelo segundo iterável. Podemos passar qualquer número de tuplas para a função chain().

```
import itertools

tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)

# using itertools
tuple3 = tuple(item for item in itertools.chain(tuple1, tuple2))
print(tuple3)
```

```
# Output (1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

Como visto na saída acima, podemos concatenar qualquer número de tuplas usando o método acima e é mais eficiente em termos de tempo do que outros métodos.

Tuplas aninhadas

Tuplas aninhadas são tuplas dentro de uma tupla, ou seja, quando uma tupla contém outra tupla como seu membro, ela é chamada de tupla aninhada.

Para recuperar os itens da tupla interna, precisamos de um loop for aninhado

```
nested_tuple = ((20, 40, 60), (10, 30, 50), "Python")

# access the first item of the third tuple
print(nested_tuple[2][0])  # P

# iterate a nested tuple
for i in nested_tuple:
    print("tuple", i, "elements")
    for j in i:
        print(j, end=", ")
    print("\n")
```

Saída

```
P

tupla (20, 40, 60) itens

20, 40, 60,



tupla (10, 30, 50) itens

10, 30, 50,



itens de tupla Python
```

```
Pitão,
```

Use funções internas com tupla

min() e max()

Como o nome sugere, a `max()`função retorna o item máximo em uma tupla e `min()`retorna o valor mínimo em uma tupla.

```
tuple1 = ('xyz', 'zara', 'abc')
# The Maximum value in a string tuple
print(max(tuple1))
# Output zara

# The minimum value in a string tuple
print(min(tuple1))
# Output abc

tuple2 = (11, 22, 10, 4)
# The Maximum value in a integer tuple
print(max(tuple2))
# Output 22
# The minimum value in a integer tuple
print(min(tuple2))
# Output 4
```

Nota : Não podemos encontrar o `max()` and `min()`para uma tupla heterogênea (tipos mistos de itens). Ele vai jogar`Type Error`

```
tuple3 = ('a', 'e', 11, 22, 15)
# max item
print(max(tuple3))
```

tudo()

No caso de `all()`função, o valor de retorno será verdadeiro somente quando todos os valores dentro forem verdadeiros. Vejamos os diferentes valores de item e os valores de retorno.

Valores de item em uma tupla	Valor de retorno
Todos os valores são verdadeiros	Verdadeiro
Um ou mais valores falsos	Falso
Todos os valores falsos	Falso
Tupla vazia	Verdadeiro

```
# all() with All True values
tuple1 = (1, 1, True)
print(all(tuple1))  # True

# all() All True values
tuple1 = (1, 1, True)
print(all(tuple1))  # True

# all() with One false value
tuple2 = (0, 1, True, 1)
print(all(tuple2))  # False

# all() with all false values
tuple3 = (0, 0, False)
print(all(tuple3))  # False

# all() Empty tuple
tuple4 = ()
print(all(tuple4))  # True
```

algum()

O método any() retornará true se houver pelo menos um valor true. No caso da tupla vazia, ela retornará false. Vamos ver a mesma combinação possível de valores para `any()`função em uma tupla e seus valores de retorno.

Valores de item em uma tupla	Valor de retorno
Todos os valores são verdadeiros	Verdadeiro
Um ou mais valores falsos	Verdadeiro
Todos os valores falsos	Falso
Tupla vazia	Falso

Da mesma forma, vamos ver cada um dos cenários acima com um pequeno exemplo.

```
# any() with All True values
tuple1 = (1, 1, True)
print(any(tuple1))  # True

# any() with One false value
tuple2 = (0, 1, True, 1)
print(any(tuple2))  # True

# any() with all false values
tuple3 = (0, 0, False)
print(any(tuple3))  # False

# any() with Empty tuple
tuple4 = ()
print(any(tuple4))  # False
```

Quando usar Tupla?

Como tuplas e listas são estruturas de dados semelhantes e ambas permitem o armazenamento sequencial de dados, as tuplas são frequentemente chamadas de listas imutáveis. Portanto, as tuplas são usadas para os seguintes requisitos em vez de listas.

- Não há `append()`ou `extend()`para adicionar itens e, da mesma forma, não há `remove()`métodos `pop()`para remover itens. Isso garante que os dados estejam protegidos contra gravação. Como as tuplas são Imutáveis, elas podem ser usadas para representar dados somente leitura ou fixos que não mudam.
- Como são imutáveis, podem ser utilizadas como chave para os dicionários, enquanto as listas não podem ser utilizadas para este fim.
- Como são imutáveis, a operação de busca é muito mais rápida que as listas. Isso ocorre porque o id dos itens permanece constante.
- As tuplas contêm dados heterogêneos (todos os tipos) que oferecem grande flexibilidade em dados que contêm combinações de tipos de dados, como caracteres alfanuméricos.