Tuple Data Structure

Anab Batool Kazmi

Tuple Data Structure

- In Python, a tuple is a data structure that is similar to a list, but with a key difference: **tuples are immutable**. This means that once you create a tuple, you cannot change its contents (add, remove, or modify elements).
- Tuples are typically used to store a collection of items **when you want the data to remain constant and unchangeable.**
- Tuple items are **ordered, unchangeable**, and **allow duplicate values**.

Create Empty Tuple

you can create an empty tuple by using

- a pair of parentheses with no elements inside.
  - empty_tuple = ()
- the tuple() constructor
  - empty_tuple = tuple()

Create Empty Tuple

Parentheses method ()

```
import sys

# create an Empty empty using paranthesis brackets []
my_tuple = ()
print("Type of my_tuple data structure is :",type(my_tuple))
print("Tuple Elements are :",my_tuple)
print("No. of Elements in my_tuple are :",len(my_tuple))
print(f"Memory reserved by my_tuple is {sys.getsizeof(my_tuple)} bytes")
print(f"Memory address of my_tuple is {id(my_tuple)}")
```

```
Type of my_tuple data structure is : <class 'tuple'>
Tuple Elements are : ()
No. of Elements in my_tuple are : 0
Memory reserved by my_tuple is 48 bytes
Memory address of my_tuple is 1561471483976
```

tuple() constructor method

```
import sys

# create an Empty empty using tuple() constructor
my_tuple = tuple()
print("Type of my_tuple data structure is :",type(my_tuple))
print("Tuple Elements are :",my_tuple)
print("No. of Elements in my_tuple are :",len(my_tuple))
print(f"Memory reserved by my_tuple is {sys.getsizeof(my_tuple)} bytes")
print(f"Memory address of my_tuple is {id(my_tuple)}")
```

```
Type of my_tuple data structure is : <class 'tuple'>
Tuple Elements are : ()
No. of Elements in my_tuple are : 0
Memory reserved by my_tuple is 48 bytes
Memory address of my_tuple is 1561471483976
```

Create Tuple With One Item

- To create **a tuple with only one item**, **you have to add a comma after the item**, otherwise Python will not recognize it as a tuple.

In [2]:
```
thistuple = ("Python",)
print(type(thistuple))

#NOT a tuple
thistuple = ("Python")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

Python - Access Tuple Items

you can access individual items (elements) in a tuple using

- indexing
  - Positive indexing (starts at 0 for the first element).
  - Negative indexing to access elements from the end of the tuple.
- Slicing
  - Slicing allows you to access a range of elements in a tuple. You can use the colon : to specify a start and end index (non-inclusive).

Python - Access Tuple Items using indexing

```python
#access tuple elements through indexing
my_tuple = ("C++", "JAVA", "C#", "JAVA", True, 1, 2)

# Access the first element
first_element = my_tuple[0]
print(f"First element of tuple is {first_element}")

# Access the third element
third_element = my_tuple[2]
print(f"Third element of tuple is {third_element}")

# Access the last element
last_element = my_tuple[-1]
print(f"Last element of tuple is {last_element}")

# Access the second-to-last element
second_to_last = my_tuple[-2]
print(f"Second to Last element of tuple is {second_to_last}")
```

```
First element of tuple is C++
Third element of tuple is C#
Last element of tuple is 2
Second to Last element of tuple is 1
```

Python - Access Tuple Items using slicing

```
#access tuple elements through slicing
my_tuple = ("C++", "JAVA", "C#", "JAVA", True, 1, 2, 3 ,8, 10)
# Get a slice of the tuple from index 1 to 3 (2nd and 3rd elements)
slice_of_tuple = my_tuple[1:3]
print(slice_of_tuple)
# Get a slice of the tuple from index -1 to -3 (last two elements)
slice_of_tuple = my_tuple[-3:-1]
print(slice_of_tuple)
# Get every second element from index 1 to 8
slice_with_step = my_tuple[1:8:2]
print(slice_with_step)
```

```
('JAVA', 'C#')
(3, 8)
('JAVA', 'JAVA', 1, 3)
```

Check if Item Exists

- To determine if a specified item is present in a tuple use the in keyword

```
#Check if Item Exists
my_tuple = ("C++", "JAVA", "C#", "JAVA", True, 1, 2, 3 ,8, 10)
if "JAVA" in my_tuple:
  print("Yes, 'JAVA' is in the  tuple")
```

```
Yes, 'JAVA' is in the  tuple
```

Python - Update Tuples

- Tuples are unchangeable, meaning **that you cannot change, add, or remove items once the tuple is created.**

- But there are some workarounds.

Python - Update Tuples-Change Tuple Values

Change Tuple Values

- Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.
- But there is a workaround. **You can convert the tuple into a list, change the list, and convert the list back into a tuple.**

```
#Convert the tuple into a list to be able to change it
my_tuple = ("C++", "JAVA", "C#", "JAVA", True, 1, 2, 3 ,8, 10)
my_list = list(my_tuple)
my_list[2] = "Data structures"
my_tuple = tuple(my_list)

print(my_tuple)
```

```
('C++', 'JAVA', 'Data structures', 'JAVA', True, 1, 2, 3, 8, 10)
```

Python - Update Tuples-Add Tuple Values

- Since tuples are immutable, they do not have a built-in append() method, but there are other ways to add items to a tuple.
- **Convert into a list**: Just like the workaround for changing a tuple, you can **convert it into a list, add your item(s), and convert it back into a tuple.**
- **Add tuple to a tuple**: You are **allowed to add tuples to tuples,** so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple by using concatenation operatior i.e. +

Python - Update Tuples-Add Tuple Values

- **Convert into a list**

```
# Convert the tuple into a list, add "Programming for AI", and convert it back into a tuple
my_tuple = ("C++", "JAVA", "C#", "JAVA", True, 1, 2, 3 ,8, 10)
my_list = list(my_tuple)
my_list.append("Programming for AI")
my_tuple = tuple(my_list)
print(my_tuple)
```

```
('C++', 'JAVA', 'C#', 'JAVA', True, 1, 2, 3, 8, 10, 'Programming for AI')
```

- **Add tuple to a tuple**

```
tuple1 = (1, 2, 3)
tuple2 = (4, )

# Concatenate tuple2 to tuple1 to create a new tuple
result_tuple = tuple1 + tuple2

print(result_tuple)  # Output: (1, 2, 3, 4)
```

```
(1, 2, 3, 4)
```

Python - Update Tuples-Add Tuple Values

Multiply Tuples

- If you want to multiply the content of a tuple a given number of times, you can use the * operator

```
Languages = ("C++", "Python")
Languages = Languages * 2

print(Languages)
```

```
('C++', 'Python', 'C++', 'Python')
```

Python - Update Tuples-Remove Item

- Tuples are unchangeable, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items

```
#Convert the tuple into a list, remove "apple", and convert it back into a tuple
my_tuple = ("C++", "JAVA", "C#", "JAVA", True, 1, 2, 3 ,8, 10)
my_list = list(my_tuple)
my_list.remove("JAVA")
my_tuple = tuple(my_list)
print(my_tuple)
```

```
('C++', 'C#', 'JAVA', True, 1, 2, 3, 8, 10)
```

Python Delete Tuple

- The **del keyword** can delete the tuple completely

```
#delete tuple completely
my_tuple = ("C++", "JAVA", "C#", "JAVA", True, 1, 2, 3 ,8, 10)
del my_tuple
print(my_tuple)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-dda91612b7d7> in <module>()
      1 my_tuple = ("C++", "JAVA", "C#", "JAVA", True, 1, 2, 3 ,8, 10)
      2 del my_tuple
----> 3 print(my_tuple)

NameError: name 'my_tuple' is not defined
```

Unpack Tuples

- Unpacking tuples in Python allows you to **assign the individual elements of a tuple to variables.**
- This is a common and useful operation when you want to work with the elements of a tuple separately.
- You can unpack a tuple using the assignment operator = and parentheses.

Unpack Tuples

```
# Creating a tuple
my_tuple = (1, 2, 3)

# Unpacking the tuple into separate variables
a, b, c = my_tuple

print(a)  # Output: 1
print(b)  # Output: 2
print(c)  # Output: 3
```

```
1
2
3
```

Unpack Tuples

- If you have more variables than there are elements in the tuple, or if you don't need all the elements, you can use **an underscore (_) as a placeholder for the elements you want to ignore**

```
# Creating a tuple
my_tuple = (10, 20, 30, 40)
# Unpacking and ignoring some elements
a, _, c, _ = my_tuple
print(a)  # Output: 10
print(c)  # Output: 30
```

```
10
30
```

```
# Creating a tuple
my_tuple = (10, 20, 30, 40,50)
# Unpacking and ignoring some elements
a, _, c, _ = my_tuple
print(a)  # Output: 10
print(c)  # Output: 30
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-884c37096721> in <module>()
      2 my_tuple = (10, 20, 30, 40,50)
      3 # Unpacking and ignoring some elements
----> 4 a, _, c, _ = my_tuple
      5 print(a)  # Output: 10
      6 print(c)  # Output: 30

ValueError: too many values to unpack (expected 4)
```

Unpack Tuples

- You can also use the *** operator** to gather the remaining elements of a **tuple into a new variable** when unpacking.
- If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

```
my_tuple = (1, 2, 3, 4, 5)

a, *rest = my_tuple

print(a)    # Output: 1
print(rest) # Output: [2, 3, 4, 5]
```

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

```
my_tuple = (1, 2, 3, 4, 5)

a, *rest, b = my_tuple

print(a)    # Output: 1
print(rest) # Output: [2, 3, 4]
print(b)    # Output: 5
```

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

Python - Loop Tuples-for loop

- You can loop through the **elements of a tuple in Python using a for loop.**

```
#loop through tuple items
my_tuple = ("C++", "JAVA", "C#", "JAVA")

for item in my_tuple:
    print(item)
```

```
C++
JAVA
C#
JAVA
```

Python - Loop Tuples-for loop

- If you also need the index of each element, you can use the **enumerate() function** along with a for loop:

```
#loop through  tuple indexes & items
my_tuple = ("C++", "JAVA", "C#", "JAVA")
for index, item in enumerate(my_tuple):
    print(f"Element at index {index}: {item}")
```

```
Element at index 0: C++
Element at index 1: JAVA
Element at index 2: C#
Element at index 3: JAVA
```

Python - Loop Tuples- for loop

- Loop Through the Index Numbers

```
my_tuple = ("C++", "JAVA", "C#", "JAVA")
for i in range(len(my_tuple)):
  print(my_tuple[i])
```

```
C++
JAVA
C#
JAVA
```

Python - Loop Tuples- While Loop

- You can loop through the tuple items by using a while loop

```
my_tuple = (100, 200, 300, 400, 500)
index = 0

while index < len(my_tuple):
    print(my_tuple[index])
    index += 1
```

```
100
200
300
400
500
```

Python Tuple count() Method

- The **count() method** returns **the number of times a specified value appears in the tuple.**

Syntax

tuple.count(*value*)

```
#count method
thistuple = ('C++', 'Python', 'C++', 'Python', 'C++', 'Python', 'C++', 'Python', 'C++', 'Python')

x = thistuple.count('C++')

print(x)
```

```
5
```

Python Tuple index() Method

- The index() method finds the first occurrence of the specified value.
- The index() method raises an exception if the value is not found.

Syntax

tuple.index(*value*)

```
thistuple = ('C++', 'Python', 'C++', 'Python', 'C++', 'Python', 'C++', 'Python', 'C++', 'Python')
x = thistuple.index('Python')

print(x)
```

```
1
```

Reference

- W3school.com