Tuples

Dr. Seema Gupta Bhol

Tuple

- Tuples are used to store multiple items in a single variable.
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
- Tuple items are ordered, unchangeable, and allow duplicate values.
- When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
- Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.
- Tuples are written with round brackets.

Tuples allow duplicate values

- thistuple = ("apple", "banana", "cherry")
  print(thistuple)
- thistuple = ("apple", "banana", "cherry", "apple", "cherry")
  print(thistuple)
- To determine how many items a tuple has, use the len() function
- Example: Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Create Tuple With One Item

- To create a tuple with only one item, we have to add a comma after the item, otherwise Python will not recognize it as a tuple.
- Example: One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))

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

A tuple can contain different data types

- tuple1 = ("apple", "banana", "cherry")
- tuple2 = (1, 5, 7, 9, 3)
- tuple3 = (True, False, False)
- tuple4 = ("abc", 34, True, 40, “Mumbai")

type()

- Tuples are defined as objects with the data type 'tuple':
- Example: Data type of a tuple is <class tuple>

```
mytuple = ("apple", "banana", "cherry")
print(type(mytuple))
```

Access Tuple Items

- We can access tuple items by referring to the index number, inside square brackets
- Example: Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

- Negative indexing start from the end.-1 refers to the last item, -2 refers to the second last item etc.
- Example :Print the last item of the tuple
- thistuple = ("apple", "banana", "cherry")
- print(thistuple[-1])

Range of Indexes

- thistuple =
  ("apple", "banana", "cherry", "orange", "kiwi", "melon", "ma
  ngo")
  print(thistuple[2:5])
- thistuple =
  ("apple", "banana", "cherry", "orange", "kiwi", "melon", "ma
  ngo")
  print(thistuple[:4])
- thistuple =
  ("apple", "banana", "cherry", "orange", "kiwi", "melon", "ma
  ngo")
  print(thistuple[2:])

Range of Negative Indexes

- Specify negative indexes if you want to start the search from the end of the tuple
- This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

print(thistuple[-4:-1])
```

Check if Item Exists (In keyword)

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

- Example : Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Change Tuple Values

- Once a tuple is created, we cannot change its values. Tuples are unchangeable, or immutable as it also is called.
- However, we can convert the tuple into a list, change the list, and convert the list back into a tuple.
- Example: Convert the tuple into a list to be able to change it

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Add tuple to a tuple

It is allowed to add tuples to tuples, so if we want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

Example : Create a new tuple with the value "orange", and add that tuple:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

Note: When creating a tuple with only one item, remember to include a comma after the item, otherwise it will not be identified as a tuple.

Remove Items

- Tuples are unchangeable, so we cannot remove items from it, but can be done by converting them into lists.

- Example :Convert the tuple into a list, remove "apple", and convert it back into a tuple

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

Unpacking a Tuple

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple:
- We are also allowed to extract the values back into variables. This is called "unpacking"
- Example: Unpacking a tuple

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)

print(yellow)

print(red)
```

- Note: The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

Note: Same with lists

Using Asterisk*

- If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

- Example: Assign the rest of the values as a list called "red":

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits
print(green)
print(yellow)
print(red)
```

- 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.

- Example :Add a list of values the "tropic" variable

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
(green, *tropic, red) = fruits
print(green)
print(tropic)
print(red)
```

Loop Through a Tuple

- It is same as lists
- Example

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Or

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

Or

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Join Two Tuples

- To join two or more tuples you can use the + operator:
- Example : Join two tuples

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

Note: Same with lists

Multiply Tuples

- If you want to multiply the content of a tuple a given number of times, you can use the * operator
- Example :Multiply the fruits tuple by 2

```
fruits = ("apple", "banana", "cherry")

mytuple = fruits * 2

print(mytuple)
```

Built-in Functions

- len(tuple): Returns the total number of items in the tuple.
- max(tuple): Returns the item from the tuple with the maximum value.
- min(tuple): Returns the item from the tuple with the minimum value.
- sum(tuple): Returns the sum of all numeric elements in the tuple.
- sorted(tuple): Returns a new sorted list of elements from the tuple.
- tuple(iterable): Converts another iterable (like a list or string) into a tuple.

Tuple Methods

- count() Returns the number of times a specified value occurs in a tuple
- index() Searches the tuple for a specified value and returns the position of where it was found

Example:

```
fruits = ('apple', ’cherry’,  'banana', 'cherry')

x = fruits.index("cherry")

print(x)

y = fruits.count("cherry")

print(y)
```

Tuple Slicing

Tuple slicing is a technique to extract a sub-part of a tuple. It uses a range of indices to create a new tuple from the original tuple.

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

# Slice from index 2 to 5

s1 = tup[2:6]

print(s1)

# Slice from the beginning to index 3

s2 = tup[:4]

print(s2)

# Slice from index 5 to the end

s3 = tup[5:]

print(s3)

# Slice the entire tuple

s4 = tup[:]

print(s4)
```

Using Negative Indices

```
tup = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
# Slice from the third last to the end
s1 = tup[-3:]
print(s1)                                   output :(7, 8, 9)
# Slice from the beginning to the third last
s2 = tup[:-3]
print(s2)                                   output (0, 1, 2, 3, 4, 5, 6)
# Slice from the third last to the second last
s3 = tup[-3:-1]                             output :(7, 8)
print(s3)
```

Using Step in Slicing

```
# Define a tuple
tup = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# Slice with a step of 2
s1 = tup[1:8:2]
print(s1)                    output: (1, 3, 5, 7)
```

A negative step value indicates that the slicing should be performed in reverse order, moving from the end of the tuple towards the beginning.

```
# Slice with a negative step (reverse the tuple)
s2 = tup[::-1]               output: (9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
print(s2)
```

Tuples vs. Lists

S.No	List	Tuple
1	Lists are mutable(can be modified).	Tuples are immutable(cannot be modified).
2	Iteration over lists is time-consuming.	Iterations over tuple is faster
3	Lists are better for performing operations, such as insertion and deletion.	Tuples are more suitable for accessing elements efficiently.
4	Lists consume more memory.	Tuples consumes less memory
5	Lists have several built-in methods.	Tuples have fewer built-in methods.
6	Lists are more prone to unexpected changes and errors.	Tuples, being immutable are less error prone.

Exercise

1. Create a tuple of 5 student marks and print average.
2. Find the largest and smallest number in a tuple.
3. Merge two tuples and sort the result.
4. Remove duplicate elements from a tuple.
5. Count how many times a number appears in a tuple.
6. Print even numbers from a tuple
7. Sort a tuple
8. Find second largest element in a tuple
9. Check if a tuple is palindrome.
10. Reverse a tuple.