

## Python Collections (Arrays)

- There are four collection data types in the Python programming language:
- List is a collection which is ordered and changeable. Allows duplicate members. []
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members. ()
- Set is a collection which is unordered and unindexed. No duplicate members. {}
- Dictionary is a collection which is unordered, changeable and indexed. No duplicate members. {}

```
In [1]: tuple1 = (1,2,3,4,5)
        tuple1[:3]
```

```
Out[1]: (1, 2, 3)
```

```
In [2]: a_tuple = (1,1,2,2,2,3,3,3,3,4,4,5, 'Sample', 'Python', 'Pyspark')
        print(type(a_tuple))
        print(a_tuple)

<class 'tuple'>
(1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 'Sample', 'Python', 'Pyspark')
```

```
In [3]: a_tuple[0:10]
```

```
Out[3]: (1, 1, 2, 2, 2, 3, 3, 3, 3, 4)
```

```
In [4]: help(a_tuple)
```

Help on tuple object:

```
class tuple(object)
| tuple(iterable=(), /)
|
| Built-in immutable sequence.
|
| If no argument is given, the constructor returns an empty tuple.
| If iterable is specified the tuple is initialized from iterable's items.
|
| If the argument is a tuple, the return value is the same object.
|
| Built-in subclasses:
|     asyncgen_hooks
|     UnraisableHookArgs
|
| Methods defined here:
|
| __add__(self, value, /)
|     Return self+value.
|
| __contains__(self, key, /)
|     Return key in self.
|
| __eq__(self, value, /)
|     Return self==value.
|
| __ge__(self, value, /)
|     Return self>=value.
|
| __getattr__(self, name, /)
|     Return getattr(self, name).
|
| __getitem__(self, key, /)
|     Return self[key].
|
| __getnewargs__(self, /)
|
| __gt__(self, value, /)
|     Return self>value.
|
| __hash__(self, /)
|     Return hash(self).
|
| __iter__(self, /)
|     Implement iter(self).
|
| __le__(self, value, /)
|     Return self<=value.
|
| __len__(self, /)
|     Return len(self).
|
| __lt__(self, value, /)
|     Return self<value.
|
| __mul__(self, value, /)
|     Return self*value.
|
| __ne__(self, value, /)
|     Return self!=value.
|
| __repr__(self, /)
|     Return repr(self).
```

```

|   __rmul__(self, value, /)
|       Return value*self.
|
|   count(self, value, /)
|       Return number of occurrences of value.
|
|   index(self, value, start=0, stop=9223372036854775807, /)
|       Return first index of value.
|
|       Raises ValueError if the value is not present.
|
|   -----
|   Class methods defined here:
|
|   __class_getitem__(...) from builtins.type
|       See PEP 585
|
|   -----
|   Static methods defined here:
|
|   __new__(*args, **kwargs) from builtins.type
|       Create and return a new object.  See help(type) for accurate signature.

```

```
In [5]: a_tuple.count(5)
```

```
Out[5]: 1
```

## Slicing

- If you omit the first index, the slice starts at the beginning. If you omit the second, the slice goes to the end. So if you omit both, the slice is a copy of the whole list.

```
In [6]: mytuple = ("apple", "banana", "cherry","kiwi","mango","orange","dragen")
print(mytuple)
print(mytuple[:4])
```

```
('apple', 'banana', 'cherry', 'kiwi', 'mango', 'orange', 'dragen')
('apple', 'banana', 'cherry', 'kiwi')
```

```
In [7]: mytuple = ("apple", "banana", "cherry","kiwi","mango","orange","dragen")
print(mytuple)
print(mytuple[3:])
```

```
('apple', 'banana', 'cherry', 'kiwi', 'mango', 'orange', 'dragen')
('kiwi', 'mango', 'orange', 'dragen')
```

```
In [8]: mytuple = ("apple", "banana", "cherry")
print('Tuple Values: ',mytuple)
print('Tuple values by index value 1: ',mytuple[0])
```

```
Tuple Values:  ('apple', 'banana', 'cherry')
Tuple values by index value 1:  apple
```

## Negative Indexing

- Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

```
In [9]: mytuple = ("apple", "banana", "cherry")
print('Negative Index -1 Value: ', mytuple[-1])
```

Negative Index -1 Value: cherry

## Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

```
In [10]: thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])

('cherry', 'orange', 'kiwi')
```

```
In [11]: thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-5:-3])

('cherry', 'orange')
```

## 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.

```
In [12]: tuple_a = (1, 2, 3, 4, 5)
list_a = list(tuple_a)
list_a[1] = 44
tuple_a = tuple(list_a)
tuple_a
```

Out[12]: (1, 44, 3, 4, 5)

```
In [13]: tuple_a
```

Out[13]: (1, 44, 3, 4, 5)

```
In [14]: x = ("apple", "banana", "cherry")
x = list(x) # Converting TUPLE into LIST
x[2] = "kiwi" # Updating a value based index in LIST
x = tuple(x) # Converting back to TUPLE From LIST
print(x)
print('X type is : ', type(x))
```

('apple', 'banana', 'kiwi')  
X type is : <class 'tuple'>

```
In [16]: type(x)
y = list(x)
print(y)
```

['apple', 'banana', 'kiwi']

## Loop Through a Tuple

- You can loop through the tuple items by using a for loop.

```
In [17]: thistuple = ("apple", "banana", "cherry")
for x in thistuple:
    print(x)
```

```
apple
banana
cherry
```

- Check if Item Exists
- To determine if a specified item is present in a tuple use the in keyword:

```
In [18]: val="bananas"
thistuple = ("apple", "banana", "cherry")
if val in thistuple:
    print("Yes, 'apple' is in the fruits tuple")
else:
    print("No, This Item is not available in this tuple")
```

```
No, This Item is not available in this tuple
```

- Tuple Length
- To determine how many items a tuple has, use the len() method

```
In [19]: thistuple = ("apple", "banana", "cherry",1,2,3, '')
print(len(thistuple))
```

```
7
```

```
In [20]: var=55
print(var)
del var
```

```
55
```

```
In [21]: print(var)
```

```
-----
NameError                                Traceback (most recent call last)
Input In [21], in <cell line: 1>()
----> 1 print(var)

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

- Remove Items
- Tuples are unchangeable, so you cannot remove items from it, but you can delete the tuple completely
- The del keyword can delete the tuple completely

```
In [22]: thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error NameError: name 'thistuple' is not defined
```

```
-----
NameError                                Traceback (most recent call last)
Input In [22], in <cell line: 3>()
      1 thistuple = ("apple", "banana", "cherry")
      2 del thistuple
----> 3 print(thistuple)

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

- Join Two Tuples
- To join two or more tuples you can use the + operator

```
In [23]: tuple1 = ("a", "b", "c", "d") # its similar to append option. just it will add end of th
tuple2 = ("e", "f", "g")
tuple3 = tuple1 + tuple2
print(tuple3)
type(tuple3)

('a', 'b', 'c', 'd', 'e', 'f', 'g')
Out[23]: tuple
```

### Slicing the Tuple

- Slicing a tuple is similar to slicing a list.

```
In [24]: tuple1 = ('Python', 'Julia', 1, 3.1415)
tuple1[1:3]

Out[24]: ('Julia', 1)
```

### The tuple() Constructor

- It is also possible to use the tuple() constructor to make a tuple.

```
In [25]: a=(12,34,45,45,56,56)
print(type(a))

<class 'tuple'>

In [26]: thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
type(thistuple)

('apple', 'banana', 'cherry')
Out[26]: tuple
```

### Nested Tuples

- It is also possible to create a tuple of tuples or tuple of lists.

```
In [27]: list1 = ['Python', 'pyspark', 1, 3.1415]
list2 = [('a', 'b'), ('c', 'd')] # List of tuples is possible too!
tuple1 = (1, 2, 3, 4, 5)
tuple2 = tuple(list1 + list2) + tuple1 # Concatenating the list and converting to tuple.
#Then adding two tuples and appening it in another tuple
tuple2

Out[27]: ('Python', 'pyspark', 1, 3.1415, ('a', 'b'), ('c', 'd'), 1, 2, 3, 4, 5)
```

In [ ]: