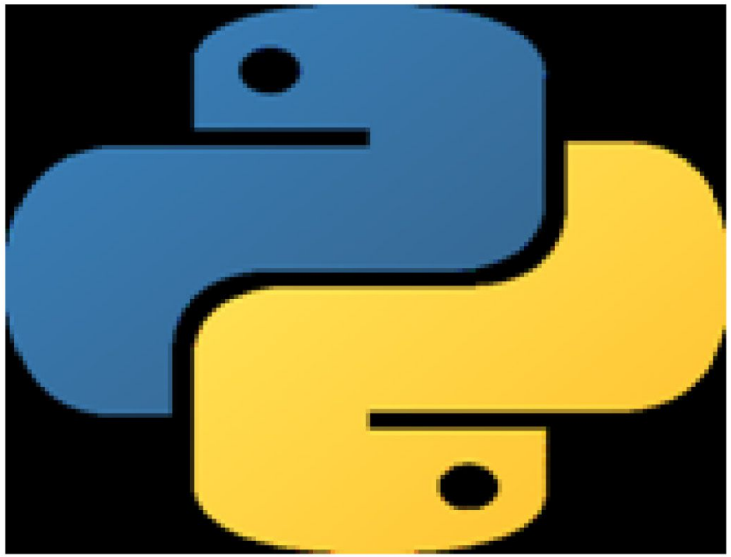


Programming With Python

3. Tuples

Tuples

Tuple is an **immutable and ordered** sequence of zero or more object references.

```
friends_tuple = ("Akhil", "James", "Amir" )

numbers_tuple = ( 1, 2, 3 )

mixed_tuple = ( "BLR", 36.5, "HYD", 42.5, "DEL", 45.2 )
```

-3	-2	-1
Akhil	James	Amir
0	1	2

-3	-2	-1
1	2	3
0	1	2

Create a tuple

Three ways:

1. Using direct initialization
2. Using tuple function
3. Converting other data type into tuple (Ex: List to Tuple)

```
# 1. Create as a constant tuple
>>> movies = ( "Ice Age", "Gravity", "Rio" )
>>> print(movies)
["Ice Age", "Gravity", "Rio"]
```

Ice Age	Gravity	Rio
0	1	2

Create a tuple of movies…

```
# 2. Create a tuple using tuple function
# Create an empty tuple
>>> movies = tuple()
>>>
print(movies) ()
>>> chars = tuple("abc")  #String to tuple of chars
>>> print(chars)
"a", "b", "c"
```

```
# Can we append values to an empty tuple???
>>> movies.append("Ice Age")
>>> movies.append("Gravity")
>>> movies.append("Rio")
>>> print(movies)
['Ice Age', 'Gravity', 'Rio']    Correct???????
```

What is the data type of a tuple?

Is it required to declare the data type of a tuple?

No

"identifiers are simply names that refer to a data object *of some type*."

- Python needs to know that you need a tuple,
- you have given a name, and
- some data items are stored in it.

```
# Can we have different types of elements in a tuple?
>>> movies = ("Ice Age", 1, "Gravity", "Rio")
>>> print(movies)
("Ice Age", 1, "Gravity", "Rio")
```

Yes

Accessing elements of a tuple (Indexing)

Printing a value using its index

```
>>> print(movies[0])
"Ice Age"
>>> print(movies[1])
"Gravity"
>>> print(movies[2])
"Gravity"
```

Out of bounds

```
>>> print(movies[5])
Index Error: tuple index out of range
```

Negative Indexing

Printing a value using negative index

```
>>> print(movies)
("Ice Age", "Gravity", "Rio")
>>> print(movies[-1])
"Rio"
>>> print(movies[-2])
"Gravity"
>>> print(movies[-3])
"Ice Age"
```

Index → -1 refers to the last item

Slicing of Tuples

Accessing a part of tuple using index range

```
>>> hair = ("black", "brown", "blonde, "red")
>>> print(hair[0:2])  #Print hair[0], hair[1]
("black", "brown")
```

```
>>> print(hair[-3:])  #Print hair[0], hair[1]
('brown', 'blonde', 'red')
```

```
>>> print(hair[1:])
["brown", "blonde", "red"]
```

```
>>> print(hair[:3])  #Print hair[0], hair[1]
["black", "brown", "blonde"]
```

hair[m : n] → Print the elements from the index "m"to "n-1"

Properties of a tuple

1. **Heterogeneous** (any data type!)
2. Stored in **Contiguous**
3. Have **random access** to any element (Using index)
4. **Ordered** (numbered from 0 to n-1)
5. Python tuples are **immutable sequences** of arbitrary objects

Intrinsic Methods of Tuples

Method	Description
tuple.index(elem)	Return index of the left most occurrence of "elem"
tuple.count(elem)	Return the number of items that is equal to elem

(a) Get the index of a given element (First occurrence..)

```
>>> movies = ("Ice Age", "Gravity", "Rio")
>>> print(movies.index("Ice Age"))
0
```

(b) Count the frequency of an element

```
>>> movies = ("Ice Age", "Gravity", "Rio", "Ice Age")
>>> print(movies.count("Ice Age"))
2
```

Delete a Tuple

(c) Delete a tuple

```
>>> movies = ["Ice Age", "Gravity", "Rio"]
>>> del movies[1]     //Deleting a single elem??? NO
>>> del movies[1:3]   // Deleting using slice??? NO
>>> del movies        // Delete completely - Possible
>>> print(movies)
NameError: name "movies" is not defined
```

Why cant we delete the individual elements of a tuple?

Tuples are immutable… We cannot change the content

We can also use the following functions:

- **min, max, len**

Tuple Membership

Check whether an element is a member of a tuple or not

- in
- not in

```
>>> a = (1, 2, 3)
>>> print(1 in a)
True
>>> print(1 not in a)
False
```

Tuple Operations (+ , *)

Two operators:

- + To concatenate two tuples
- * Repeats a tuple a given number of times

```
# Concatenation
>>> a = (1, 2, 3)
>>> b = (4, 5, 6)
>>> a + b
(1, 2, 3, 4, 5, 6)
```

```
# Repeat
>>> a = (1, 2, 3)
>>> a * 2
(1, 2, 3, 1, 2, 3)
```

Tuples and for loop

How to use for loop to parse a tuple?

```
for var in Tuple:
    BODY of the LOOP
```

Example:

```
>>> friends = ("Ram", "Rahim", "John")
>>> for friend in friends:
        print(friend)
```

Output:

```
Ram
Rahim
John
```

For (every) friend in (the tuple of) friends names, print (the name of the) friend.

Enumerate a Tuple

enumerate generates pairs of both (index, value) during the tuple traversal.

Example:

```
>>> friends = ("Ram", "Rahim", "John")
>>> for (i, v) in enumerate(friends):
            print(i, v)
(0, 'Ram')
(1, 'Rahim')
(2, 'John')
```

It generates key value pairs for the members of a tuple.

Nested Tuple

A tuple can be an element of another tuple

Example:

```
>>> nested = ("Ram", "CDAC", (1, 2, 3))
>>>
print(nested[2])
(1, 2, 3)
```

How to get the element from the nested tuple?

```
>>> n = nested[2]
>>> print(n[0])
1
```

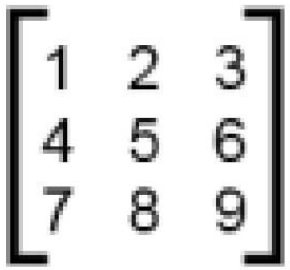
OR

```
>>> nested[2][0]
1
```

Matrix as a nested tuple

A matrix can be represented using a nested tuple.

Example:




```
>>> mx = ((1, 2, 3), (4, 5, 6), (7, 8, 9))
>>> print(mx[0])
(1, 2, 3)
>>>
print(mx[1][2]) 6
```

How to get the element from the nested tuple?

Tuple Comprehension

Tuple comprehensions provide a concise way to create new tuples.

```
>>> squares = tuple(map(lambda x: x**2, range(5)))
                    (OR)
>>> squares = tuple(x**2 for x in range(5))
```

Example1 – Calculate the prime numbers

Calculate the prime numbers between 1 to 100

```
>>> not_prime_tuple = tuple(j  for i in range(2, 8) for
j in range(i*2, 100, i))


>>> primes = tuple(x  for x in range(2, 100) if x not
in not_prime_tuple)


>>> print(primes)
```

Output:

```
(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97)
```

Example2 – Celsius to Fahrenheit

Convert Celsius to Fahrenheit

```
>>> Celsius = (39.2, 36.5, 37.3, 37.8)

>>> Fahrenheit = tuple( ((float(9)/5)*x + 32) for x in
Celsius )

>>> print(Fahrenheit)
```

```
Output:
[102.56,     97.700000000000003,     99.140000000000001,
100.03999999999999]
```

Example3 – Transpose using zip

Find Transpose of a matrix:

```
>>> tuple(zip(*zipped))
```

Output:

```
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```

Note: * - operator to unpack the arguments out of a tuple or tuple:

Thank You