In Python, **tuples** are part of the standard language. This is a data structure very similar to the **list** data structure. The main difference being that tuple manipulation are faster than list because tuples are immutable.

Constructing tuples

To create a tuple, place values within brackets:

```
>>> l = (1, 2, 3)
>>> l[0]
1
```

It is also possible to create a tuple without parentheses, by using commas:

```
>>> l = 1, 2
>>> l
(1, 2)
```

If you want to create a tuple with a single element, you must use the comma:

```
>>> singleton = (1, )
```

You can repeat a tuples by multiplying a tuple by a number:

```
>>> (1,) * 5
(1, 1, 1, 1, 1)
```

Note that you can concatenate tuples and use augmented assignement (*=, +=):

```
>>> s1 = (1,0)
>>> s1 += (1,)
>>> s1
(1, 0, 1)
```

Tuple methods

Tuples are optimised, which makes them very simple objects. There are two methods available only:

- *index, to find occurence of a value*
- *count, to count the number of occurence of a value*

```
>>> l = (1,2,3,1)
>>> l.count(1)
2
>>> l.index(2)
1
```

Interests of tuples

So, Tuples are useful because there are

- *faster than lists*
- *protect the data, which is immutable*
- *tuples can be used as keys on dictionaries*

In addition, it can be used in different useful ways:

Tuples as key/value pairs to build dictionaries

```
>>> d = dict([('jan', 1), ('feb', 2), ('march', 3)])
>>> d['feb']
2
```

4.2.3.2. signing multiple values

```
>>> (x,y,z) = ('a','b','c')
>>> x
'a'
>>> (x,y,z) = range(3)
>>> x
0
```

Tuple Unpacking

Tuple unpacking allows to extract tuple elements automatically is the list of variables on the left has the same number of elements as the length of the tuple

```
>>> data  = (1,2,3)
>>> x, y, z = data
>>> x
1
```

Tuple can be use as swap function

This code reverses the contents of 2 variables x and y:

```
>>> (x,y) = (y,x)
```

Warning

Consider the following function:

```
def swap(a, b):
    (b, a) = (a, b)
```

then:

```
a = 2
b = 3
swap(a, b)
#a is still 2 and b still 3 !! a and b are indeed passed
by value not reference.
```

Misc

length

To find the length of a tuple, you can use the `len()` function:

```
>>> t= (1,2)
>>> len(t)
2
```

Slicing (extracting a segment)

```
>>> t = (1,2,3,4,5)
>>> t[2:]
(3, 4, 5)
```

Copy a tuple

To copy a tuple, just use the assignement:

```
>>> t = (1, 2, 3, 4, 5)
>>> newt = t
>>> t[0] = 5
>>> newt
(1, 2, 3, 4, 5)
```

Warning

You cannot copy a list with the = sign because lists are mutables. The = sign creates a reference not a copy. Tuples are immutable therefore a = sign does not create a reference but a copy as expected.

Tuple are not fully immutable !!

If a value within a tuple is mutable, then you can change it:

```
>>> t = (1, 2, [3, 10])
>>> t[2][0] = 9
>>> t
(1, 2, [9, 10])
```

Convert a tuple to a string

You can convert a tuple to a string with either:

```
>>> str(t)
```

or

```
>>> `t`
```

math and comparison

comparison operators and mathematical functions can be used on tuples. Here are some examples:

```
>>> t = (1, 2, 3)
>>> max(t)
3
```