Tuples

- A tuple is similar to list
- Duplicates allowed
- The difference between a tuple and a list is that we **can't change the elements of tuple once it is assigned/created** whereas in the **list, elements can be changed.**
- Tuples are **immuatable** data structures whereas list is **mutable**.

> **Mutable:** Can be changed or updated after creation
> **Immutable:** can't be changed or updated

Tuple creation

In [2]:
```
# empty tuple
t = ()

# tuple having integers
t = (1, 2, 3)
print(t)

# tuple with mixed datatypes
t = (1, 'iota', 28, 'academy')
print(t)

# nested tuple
t = (2, (2, 3, 4), [1, 'raju', 28, 'abc'],{"key":'value'})
print(t)
```

```
(1, 2, 3)
(1, 'iota', 28, 'academy')
(2, (2, 3, 4), [1, 'raju', 28, 'abc'], {'key': 'value'})
```

In [3]:
```
# only parenthesis is not enough   # boundary case between tuple and string/numbers
t =(1)
print(type(t))   # data type of t is int and not tuple
```

```
<class 'int'>
```

In [4]:
```
#need a comma at the end
t = ("iota",)
type(t)
```

Out[4]:
```
tuple
```

In [5]:
```
tup = (100+200+300
       +500+600)

tup1 = (100+200+300
       +500+600,)

#notice comma at the end inside parenthesis
print(type(tup))
print(type(tup1))
```

```
<class 'int'>
<class 'tuple'>
```

In [6]:
```
# parenthesis is optional
i = "iota","academy"
print(type(i))

print(i)
```

```
<class 'tuple'>
('iota', 'academy')
```

Accessing Elements in Tuple

In [8]:
```
# similar to accessing elements in list

# Syntax: variable_name[index]

t = ('iota', 'academy', 'foz', 'learning', 'python')

print(t[1])
```

```
academy
```

In [9]:
```
# supports negative index
print(t[-3]) # print last element in a tuple
```

```
foz
```

In [10]:
```
# nested tuple
t = (100,'iota', ('academy', 'foz', 'learning', 'python'),(1,'iota'))

print(t[2][0])
```

```
academy
```

In [11]:
```
print(t[2][-1]) # observe output
```

```
python
```

In [12]:
```
# Slicing
t = (1, 2, 3, 4, 5, 6)

print(t[1:4]) # variable_name(start:end:step)

# print elements from starting to 2nd last elements
print(t[:-2])

# print elements from starting to end
print(t[::-1])

print(t)
```

```
(2, 3, 4)
(1, 2, 3, 4)
(6, 5, 4, 3, 2, 1)
(1, 2, 3, 4, 5, 6)
```

Changing a Tuple

- unlike lists, tuples are immutable
- This means that elements of a tuple cannot be changed once it has been assigned.

> Note: if the element is itself a **mutable datatype like list**, its nested items can be changed.

In [13]:
```
# creating tuple
t = (1, 2, 3, 4, [5, 6, 7],{"abc":100},(1,2,3,5))

print(len(t))

t[2] = 'x' # will get TypeError
```

```
7
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 6
      2 t = (1, 2, 3, 4, [5, 6, 7],{"abc":100},(1,2,3,5))
      4 print(len(t))
----> 6 t[2] = 'x'

TypeError: 'tuple' object does not support item assignment
```

In [14]:
```
t[4][1] = 'iota'
print(t)
len(t)
```

```
(1, 2, 3, 4, [5, 'iota', 7], {'abc': 100}, (1, 2, 3, 5))
```

Out[14]:
```
7
```

In [15]:
```
# updating list inside tuple

t[4].append("Hello")
print(len(t))
t
```

```
7
```

Out[15]:
```
(1, 2, 3, 4, [5, 'iota', 7, 'Hello'], {'abc': 100}, (1, 2, 3, 5))
```

In [16]:
```
# Concatinating tuples
t1 = (1, 2, 3)
t2 = (4, 5, 6)
t1 =  t1 + t2 # (1,2,3,4,5,6)
print(t1)
```

```
(1, 2, 3, 4, 5, 6)
```

In [17]:
```
# repeat the elements in a tuple for a given number of times using the * operator.
t = ('iota','academy') * 4
print(t)
```

```
('iota', 'academy', 'iota', 'academy', 'iota', 'academy', 'iota', 'academy')
```

Tuple Deletion

In [18]:
```
#we cannot change the elements in a tuple.
# That also means we cannot delete or remove items from a tuple.

#delete entire tuple using del keyword
t = (1, 2, 3, 4, 5, 6)

#delete entire tuple
del t
```

Tuple Methods

Method	Description
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

Tuple Count

- Returns the number of times a specified value occurs in a tuple

In [17]:
```
t = (1, 2, 3, 1, 3, 3, 4, 1)

#get the frequency of particular element appears in a tuple
t.count(1)
```

Out[17]:
```
3
```

Tuple Index

- Searches the tuple for a specified value and returns the position of where it was found
- Return first index of value.
- Raises ValueError if the value is not present.

In [19]:
```
t = (1, 2, 3, 1, 34,3, 3, 4, 1)

t.index(34)
```

```
4
```

In [22]:
```
# Example
t = (1, 2, 3, 1, 34,3, 3, 4, 1)


print(t.index(3) + t.index(3)) # return index of the first element is equal to 3
```

```
4
```

Tuple Memebership

In [23]:
```
# test if an item exists in a tuple or not, using the keyword in.
t = (1, 2, 3, 4, 5, 6)

print(1 in t)
```

```
True
```

In [24]:
```
print(7 not in t)
```

```
True
```

Built in/ Standard Functions

Tuple Length

In [29]:
```
t = (1, 2, 3, 4, 5, 6)
print(len(t))
```

```
6
```

Tuple Sort

In [30]:
```
t = (4, 5, 1, 2, 3)
t1 = sorted(t)
print(t1)  # return a sorted list
```

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

In [31]:
```
t = (4, 5, 1, 2, 3)

t1 = tuple(sorted(t))
print(t)
print(t1)
```

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

In [32]:
```
t = (4, 5, 1, 2, 3)

new_t = tuple(sorted(t))
print(new_t) # Take elements in the tuple and return a new sorted list
             # (does not sort the tuple itself).

print(t)
```

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

max()

In [33]:
```
# get the largest element in a tuple
t = (2, 5, 1, 6)

print(max(t))
```

```
6
```

min()

In [34]:
```
# get the smallest element in a tuple
print(min(t))
```

```
1
```

sum()

In [26]:
```
# get sum of elments in the tuple
print(sum(t))
```

```
14
```

That's Great!