Tuples

A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

In short, a tuple is an immutable list. A tuple can not be changed in any way once it is created.

Characterstics

- Ordered
- Unchangeble
- Allows duplicate

Plan of attack

- Creating a Tuple
- Accessing items
- Editing items
- Adding items
- Deleting items
- Operations on Tuples
- Tuple Functions

Creating Tuples

```
# empty
t1 = ()
print(t1)
# create a tuple with a single item
t2 = ('hello',)
print(t2)
print(type(t2))
# homo
t3 = (1,2,3,4)
print(t3)
# hetro
t4 = (1,2.5,True,[1,2,3])
print(t4)
# tuple
t5 = (1,2,3,(4,5))
print(t5)
# using type conversion
```

```
t6 = tuple('hello')
print(t6)
```

```
()
('hello',)
<class 'tuple'>
(1, 2, 3, 4)
(1, 2.5, True, [1, 2, 3])
(1, 2, 3, (4, 5))
('h', 'e', 'l', 'l', 'o')
```

Accessing Items

- Indexing
- Slicing

```
print(t3)
print(t3[0])
print(t3[-1])
```

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

```
t5[-1][0]
```

```
4
```

Editing items

```
print(t3)
t3[0] = 100
# immutable just like strings
```

```
(1, 2, 3, 4)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-49d9e1416ccf> in <module>
      1 print(t3)
----> 2 t3[0] = 100

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

SEARCH STACK OVERFLOW

Adding items

```
print(t3)
# not possible
```

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

Deleting items

```
print(t3)
del t3
print(t3)
```

```
(1, 2, 3, 4)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-0a67b29ad777> in <module>
      1 print(t3)
      2 del t3
----> 3 print(t3)

NameError: name 't3' is not defined
```

SEARCH STACK OVERFLOW

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

t[-1:-4:-1]
```

```
(5, 4, 3)
```

```
print(t5)
del t5[-1]
```

```
(1, 2, 3, (4, 5))
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-2b39d140e8ae> in <module>
      1 print(t5)
----> 2 del t5[-1]

TypeError: 'tuple' object doesn't support item deletion
```

SEARCH STACK OVERFLOW

Operations on Tuples

```
# + and *
t1 = (1,2,3,4)
t2 = (5,6,7,8)

print(t1 + t2)

print(t1*3)
# membership
1 in t1
# iteration
for i in t1:
  print(i)
```

```
(1, 2, 3, 4, 5, 6, 7, 8)
(1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)
1
2
3
4
```

Tuple Functions

```
# len/sum/min/max/sorted
t = (1,2,3,4)
len(t)

sum(t)

min(t)

max(t)

sorted(t,reverse=True)
```

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

```
# count

t = (1,2,3,4,5)

t.count(50)
```

```
0
```

```
# index
t.index(50)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-51-cae2b6ba49a8> in <module>
      1 # index
----> 2 t.index(50)

ValueError: tuple.index(x): x not in tuple
```

SEARCH STACK OVERFLOW

Difference between Lists and Tuples

- Syntax
- Mutability
- Speed
- Memory
- Built in functionality
- Error prone
- Usability

```
import time

L = list(range(100000000))
T = tuple(range(100000000))

start = time.time()
for i in L:
  i*5
print('List time',time.time()-start)

start = time.time()
for i in T:
  i*5
print('Tuple time',time.time()-start)
```

```
List time 9.853569507598877
Tuple time 8.347511053085327
```

```
import sys

L = list(range(1000))
T = tuple(range(1000))

print('List size',sys.getsizeof(L))
print('Tuple size',sys.getsizeof(T))
```

```
List size 9120
Tuple size 8056
```

```
a = [1,2,3]
b = a

a.append(4)
print(a)
print(b)
```

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

```
a = (1,2,3)
b = a

a = a + (4,)
print(a)
print(b)
```

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

Why use tuple?

Special Syntax

```
# tuple unpacking
a,b,c = (1,2,3)
print(a,b,c)
```

```
1 2 3
```

```
a,b = (1,2,3)
print(a,b)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-55-22f327f11d4b> in <module>
----> 1 a,b = (1,2,3)
      2 print(a,b)

ValueError: too many values to unpack (expected 2)

SEARCH STACK OVERFLOW
```

```
a = 1
b = 2
```

```
a,b = b,a

print(a,b)
```

```
2 1
```

```
a,b,*others = (1,2,3,4)
print(a,b)
print(others)
```

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

```
# zipping tuples
a = (1,2,3,4)
b = (5,6,7,8)

tuple(zip(a,b))
```

```
((1, 5), (2, 6), (3, 7), (4, 8))
```

Sets

A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).

However, a set itself is mutable. We can add or remove items from it.

Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

Characterstics:

- Unordered
- Mutable
- No Duplicates
- Can't contain mutable data types

Creating Sets

```
# empty
s = set()
print(s)
print(type(s))
# 1D and 2D
```

```
s1 = {1,2,3}
print(s1)
#s2 = {1,2,3,{4,5}}
#print(s2)
# homo and hetro
s3 = {1,'hello',4.5,(1,2,3)}
print(s3)
# using type conversion

s4 = set([1,2,3])
print(s4)
# duplicates not allowed
s5 = {1,1,2,2,3,3}
print(s5)
# set can't have mutable items
s6 = {1,2,[3,4]}
print(s6)
```

```
set()
<class 'set'>
{1, 2, 3}
{1, 4.5, (1, 2, 3), 'hello'}
{1, 2, 3}
{1, 2, 3}
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-71-ab3c7dde6aed> in <module>
     19 print(s5)
     20 # set can't have mutable items
---> 21 s6 = {1,2,[3,4]}
     22 print(s6)

TypeError: unhashable type: 'list'
```

SEARCH STACK OVERFLOW

```
s1 = {1,2,3}
s2 = {3,2,1}

print(s1 == s2)
```

```
True
```

Accessing Items

```
s1 = {1,2,3,4}
```

```
s1[0:3]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-4c49b6b6050d> in <module>
      1 s1 = {1,2,3,4}
----> 2 s1[0:3]

TypeError: 'set' object is not subscriptable
```

SEARCH STACK OVERFLOW

Editing Items

```
s1 = {1,2,3,4}
s1[0] = 100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-76-bd617ce25076> in <module>
      1 s1 = {1,2,3,4}
----> 2 s1[0] = 100

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

SEARCH STACK OVERFLOW

Adding Items

```
S = {1,2,3,4}
# add
# S.add(5)
# print(S)
# update
S.update([5,6,7])
print(S)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

Deleting Items

```
# del
s = {1,2,3,4,5}
# print(s)
# del s[0]
# print(s)
```

```
# discard
# s.discard(50)
# print(s)
# remove
# s.remove(50)
# print(s)
# pop
# s.pop()
# clear
s.clear()
print(s)
```

```
set()
```

Set Operation

```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
s1 | s2
# Union(|)
# Intersection(&)
s1 & s2
# Difference(-)
s1 - s2
s2 - s1
# Symmetric Difference(^)
s1 ^ s2
# Membership Test
1 not in s1
# Iteration
for i in s1:
  print(i)
```

```
1
2
3
4
5
```

Set Functions

```
# len/sum/min/max/sorted
s = {3,1,4,5,2,7}
len(s)
sum(s)
min(s)
```

```
max(s)
sorted(s,reverse=True)
```

```
[7, 5, 4, 3, 2, 1]
```

```
# union/update
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}

# s1 | s2
s1.union(s1)

s1.update(s2)
print(s1)
print(s2)
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
{4, 5, 6, 7, 8}
```

```
# intersection/intersection_update
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}

s1.intersection(s2)

s1.intersection_update(s2)
print(s1)
print(s2)
```

```
{4, 5}
{4, 5, 6, 7, 8}
```

```
# difference/difference_update
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}

s1.difference(s2)

s1.difference_update(s2)
print(s1)
print(s2)
```

```
{1, 2, 3}
{4, 5, 6, 7, 8}
```

```
# symmetric_difference/symmetric_difference_update
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}

s1.symmetric_difference(s2)
```

```
s1.symmetric_difference_update(s2)
print(s1)
print(s2)
```

```
{1, 2, 3, 6, 7, 8}
{4, 5, 6, 7, 8}
```

```
# isdisjoint/issubset/issuperset
s1 = {1,2,3,4}
s2 = {7,8,5,6}

s1.isdisjoint(s2)
```

```
True
```

```
s1 = {1,2,3,4,5}
s2 = {3,4,5}

s1.issuperset(s2)
```

```
True
```

```
# copy
s1 = {1,2,3}
s2 = s1.copy()

print(s1)
print(s2)
```

```
{1, 2, 3}
{1, 2, 3}
```

Frozenset

Frozen set is just an immutable version of a Python set object

```
# create frozenset
fs1 = frozenset([1,2,3])
fs2 = frozenset([3,4,5])

fs1 | fs2
```

```
frozenset({1, 2, 3, 4, 5})
```

```
# what works and what does not
# works -> all read functions
# does't work -> write operations
```

```
# When to use
# 2D sets
fs = frozenset([1,2,frozenset([3,4])])
fs
```

```
frozenset({1, 2, frozenset({3, 4})})
```

Set Comprehension

```
# examples

{i**2 for i in range(1,11) if i>5}
```

```
{36, 49, 64, 81, 100}
```

Dictionary

Dictionary in Python is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.

In some languages it is known as map or assosiative arrays.

dict = { 'name' : 'nitish' , 'age' : 33 , 'gender' : 'male' }

Characterstics:

- Mutable
- Indexing has no meaning
- keys can't be duplicated
- keys can't be mutable items

Create Dictionary

```
# empty dictionary
d = {}
d
# 1D dictionary
d1 = { 'name' : 'nitish' ,'gender' : 'male' }
d1
# with mixed keys
```

```
d2 = {(1,2,3):1,'hello':'world'}
d2
# 2D dictionary -> JSON
s = {
    'name':'nitish',
     'college':'bit',
     'sem':4,
     'subjects':{
         'dsa':50,
         'maths':67,
         'english':34
     }
}
s
# using sequence and dict function
d4 = dict([('name','nitish'),('age',32),(3,3)])
d4
# duplicate keys
d5 = {'name':'nitish','name':'rahul'}
d5
# mutable items as keys
d6 = {'name':'nitish',(1,2,3):2}
print(d6)
```

```
{'name': 'nitish', (1, 2, 3): 2}
```

Accessing items

```
my_dict = {'name': 'Jack', 'age': 26}
# []
my_dict['age']
# get
my_dict.get('age')

s['subjects']['maths']
```

```
67
```

Adding key-value pair

```
d4['gender'] = 'male'
d4
d4['weight'] = 72
d4

s['subjects']['ds'] = 75
s
```

```
{'name': 'nitish',
 'college': 'bit',
 'sem': 4,
 'subjects': {'dsa': 50, 'maths': 67, 'english': 34, 'ds': 75}}
```

Remove key-value pair

```
d = {'name': 'nitish', 'age': 32, 3: 3, 'gender': 'male', 'weight': 72}
# pop
#d.pop(3)
#print(d)
# popitem
#d.popitem()
# d.popitem()
# print(d)
# del
#del d['name']
#print(d)
# clear
d.clear()
print(d)

del s['subjects']['maths']
s
```

```
{}
{'name': 'nitish',
 'college': 'bit',
 'sem': 4,
 'subjects': {'dsa': 50, 'english': 34, 'ds': 75}}
```

Editing key-value pair

```
s['subjects']['dsa'] = 80
s
```

```
{'name': 'nitish',
 'college': 'bit',
 'sem': 5,
 'subjects': {'dsa': 80, 'english': 34, 'ds': 75}}
```

Dictionary Operations

- Membership
- Iteration

```
print(s)

'name' in s
```

```
{'name': 'nitish', 'college': 'bit', 'sem': 5, 'subjects': {'dsa': 80, 'english': 34, 'c
True
```

```
d = {'name':'nitish','gender':'male','age':33}

for i in d:
  print(i,d[i])
```

```
name nitish
gender male
age 33
```

Dictionary Functions

```
# len/sorted
len(d)
print(d)
sorted(d,reverse=True)
max(d)
```

```
{'name': 'nitish', 'gender': 'male', 'age': 33}
'name'
```

```
# items/keys/values
print(d)

print(d.items())
print(d.keys())
print(d.values())
```

```
{'name': 'nitish', 'gender': 'male', 'age': 33}
dict_items([('name', 'nitish'), ('gender', 'male'), ('age', 33)])
dict_keys(['name', 'gender', 'age'])
dict_values(['nitish', 'male', 33])
```

```
# update
d1 = {1:2,3:4,4:5}
d2 = {4:7,6:8}

d1.update(d2)
print(d1)
```

```
{1: 2, 3: 4, 4: 7, 6: 8}
```

Dictionary Comprehension


```
# print 1st 10 numbers and their squares
{i:i**2 for i in range(1,11)}
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

```
distances = {'delhi':1000,'mumbai':2000,'bangalore':3000}
print(distances.items())
```

```
dict_items([('delhi', 1000), ('mumbai', 2000), ('bangalore', 3000)])
```

```
# using existing dict
distances = {'delhi':1000,'mumbai':2000,'bangalore':3000}
{key:value*0.62 for (key,value) in distances.items()}
```

```
{'delhi': 620.0, 'mumbai': 1240.0, 'bangalore': 1860.0}
```

```
# using zip
days = ["Sunday", "Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
temp_C = [30.5,32.6,31.8,33.4,29.8,30.2,29.9]

{i:j for (i,j) in zip(days,temp_C)}
```

```
{'Sunday': 30.5,
 'Monday': 32.6,
 'Tuesday': 31.8,
 'Wednesday': 33.4,
 'Thursday': 29.8,
 'Friday': 30.2,
 'Saturday': 29.9}
```

```
# using if condition
products = {'phone':10,'laptop':0,'charger':32,'tablet':0}

{key:value for (key,value) in products.items() if value>0}
```

```
{'phone': 10, 'charger': 32}
```

```
# Nested Comprehension
# print tables of number from 2 to 4
```

```
{i:{j:i*j for j in range(1,11)} for i in range(2,5)}
```

```
{2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18, 10: 20},
 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15, 6: 18, 7: 21, 8: 24, 9: 27, 10: 30},
 4: {1: 4, 2: 8, 3: 12, 4: 16, 5: 20, 6: 24, 7: 28, 8: 32, 9: 36, 10: 40}}
```

```
{
    2:{1:2,2:4,3:6,4:8},
    3:{1:3,2:6,3:9,4:12},
    4:{1:4,2:8,3:12,4:16}
}
```