Tuples and Dictionaries

What is a Tuple?

A tuple is an ORDERED sequence of elements of different data types (int, float, string, list, even another tuple). Elements are stored in ( ) round brackets, separated by commas. Elements are accessed by index, from 0.

Creating tuples

```
>>> tuple1 = (1,2,3,4,5)   #integers
>>> tuple2 = ('Eco',87,'Acc',89.6)
>>> tuple3 = (10,20,30,[40,50]) #list in
>>> tuple4 = (1,2,3,(10,20))  #tuple in
```

A tuple can hold values of many types at once, and even nest a list or tuple inside.

We usually use a LIST for same-type data, and a TUPLE for different-type data.

The Comma Rule

A single element in a tuple MUST be followed by a comma. Without it Python sees a ~~tuple~~ plain value (int / str), NOT a tuple.

Wrong vs right

```
>>> tuples = (20)      #no comma
>>> type(tuples)
<class 'int'>          #just an int?

>>> tuples = (20,)     #comma added
>>> type(tuples)
<class 'tuple'>        #now a tuple
```

No brackets = tuple too

A sequence of comma-separated values with NO parentheses is treated as a tuple by default.

```
>>> seq = 1,2,3
>>> type(seq)
<class 'tuple'>
```

Accessing Elements

Elements* of a tuple are accessed just like a list or string - using indexing and slicing. Index starts at 0; -1 is the last element.

```
>>> tuple1 = (2,4,6,8,10,12)
>>> tuple1[0]          #first element
2
>>> tuple1[3]          #fourth element
8
>>> tuple1[1+4]        #index by expr
12
>>> tuple1[-1]         #first from right
12
>>> tuple1[15]         #out of range
IndexError: tuple index out of range
```

Two-way indexing

Positive index counts from the left (0,1,2..), negative counts from the right (-1,-2..).
An out-of-range index raises IndexError.

Tuples are Immutable

A tuple is IMMUTABLE: once created, its elements ~~can~~ canNOT be changed. An attempt to reassign an element raises a TypeError.

```
>>> tuple1 = (1,2,3,4,5)
>>> tuple1[4] = 10
TypeError: 'tuple' object does not
support item assignment
```

But a mutable element can change

If a tuple holds a LIST, the list itself can still be modified in place (the tuple slot keeps pointing to the same list object).

```
>>> tuple2 = (1,2,3,[8,9])
>>> tuple2[3][1] = 10  #change list item
>>> tuple2
(1, 2, 3, [8, 10])
```

So immutability protects the tuple's own slots, not the inside of a mutable element.

Tuple Operations : +

Concatenation (+)

The + operator joins two tuples and gives a NEW tuple. Both operands must be tuples.

```
>>> tuple1 = (1,3,5,7,9)
>>> tuple2 = (2,4,6,8,10)
>>> tuple1 + tuple2
(1, 3, 5, 7, 9, 2, 4, 6, 8, 10)
```

Extending a tuple

Since tuples are immutable, extending really builds a NEW tuple. Note the (6,) comma!

```
>>> tuple6 = (1,2,3,4,5)
>>> tuple6 = tuple6 + (6,)    #one item
>>> tuple6
(1, 2, 3, 4, 5, 6)
>>> tuple6 = tuple6 + (7,8,9) #many
>>> tuple6
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Repetition & Membership

Repetition (*)

The * operator repeats a tuple. First operand is a tuple, second must be an INTEGER.

```
>>> tuple1 = ('Hello','World')
>>> tuple1 * 3
('Hello','World','Hello','World',
 'Hello','World')
>>> tuple2 = ('Hello',)  #single elem
>>> tuple2 * 4
('Hello','Hello','Hello','Hello')
```

Membership (in / not in)

in returns True if the element is present, else False. not in does the opposite.

```
>>> tuple1 = ('Red','Green','Blue')
>>> 'Green' in tuple1
True
>>> 'Green' not in tuple1
False
```

Slicing a Tuple

Slicing pulls out a part of a tuple. The form is tuple[start : stop : step]. stop is NOT included. Default step is 1.

```
>>> t = (10,20,30,40,50,60,70,80)
>>> t[2:7]          #index 2 to 6
(30, 40, 50, 60, 70)
>>> t[:5]           #start from 0
(10, 20, 30, 40, 50)
>>> t[2:]           #till the end
(30, 40, 50, 60, 70, 80)
>>> t[0:len(t):2] #step size 2
(10, 30, 50, 70)
>>> t[-6:-4]        #negative index
(30, 40)
>>> t[::-1]         #reverse order
(80, 70, 60, 50, 40, 30, 20, 10)
```

t[::-1] is a quick way to reverse a tuple.

Tuple Methods (1)

Python gives built-in functions + methods to work on tuples. First four:

len() and tuple()

```
>>> tuple1 = (10,20,30,40,50)
>>> len(tuple1)          #count items
5
>>> tuple('aeiou')       #from a string
('a', 'e', 'i', 'o', 'u')
>>> tuple(range(5))      #from a range
(0, 1, 2, 3, 4)
```

count() and index()

```
>>> t = (10,20,30,10,40,10,50)
>>> t.count(10)   #times 10 appears
3
>>> t.index(30)   #index of first 30
2
```

Tuple Methods (2)

sorted()

Returns a NEW sorted LIST. The original tuple is left unchanged.

```
>>> t = ('Rama', 'Heena', 'Raj', 'Aditya')
>>> sorted(t)
['Aditya', 'Heena', 'Raj', 'Rama']
```

min() , max() , sum()

```
>>> t = (19,12,56,18,9,87,34)
>>> min(t)      #smallest
9
>>> max(t)      #largest
87
>>> sum(t)      #total
235
```

- sum() needs all-numeric elements.
- sorted() output is a list, not a tuple.

Tuple Assignment

A tuple of variables on the LEFT can take values from a tuple on the RIGHT. Count of names must equal count of values.

```
>>> (num1,num2) = (10,20)  #unpack
>>> num1
10
>>> record = ('Pooja',40,'CS')
>>> (name,rollNo,subject) = record
>>> name
'Pooja'
>>> (a,b,c,d) = (5,6,8)   #mismatch
ValueError: not enough values to unpack
```

Packing & unpacking

- Packing: values -> one tuple (t = 1,2,3).
- Unpacking: tuple -> many names.

Swap without a temp

```
(num1,num2) = (num2,num1)  #swap
```

Nested Tuples

A tuple inside another tuple is a NESTED tuple. Handy to store records of many items.

Program 10-1 : student records

```
st = ((101,'Aman',98),
      (102,'Geet',95),
      (103,'Sahil',87),
      (104,'Pawan',79))                    <- st[i][j]
print('S_No','Roll','Name','Marks')
for i in range(0,len(st)):                 <- = row i,
    print(i+1,st[i][0],
          st[i][1],st[i][2])               <- col j
```

Output

```
S_No  Roll  Name  Marks
1     101   Aman  98
2     102   Geet  95
3     103   Sahil 87
4     104   Pawan 79
```

Tuple Handling

Program 10-2 : swap two numbers

```
num1 = int(input('First: '))
num2 = int(input('Second: '))
(num1,num2) = (num2,num1)  #swap
print('First:',num1)
print('Second:',num2)
```

```
First: 10
Second: 5.
```

Program 10-3 : area from a function

A function can RETURN a tuple - a neat way to send back more than one value.

```
def circle(r):
    area = 3.14*r*r
    circ = 2*3.14*r
    return (area, circ)  #tuple back
area,circ = circle(5)   #unpack
```

Program 10-4 : Max & Min

Read n numbers, store in a tuple, print the largest and smallest.

```
numbers = tuple()          #empty tuple
n = int(input('How many?: '))
for i in range(0,n):
    num = int(input())
    numbers = numbers + (num,)
print('Max:', max(numbers))
print('Min:', min(numbers))
```

```
(9, 8, 10, 12, 15)
Max: 15
Min: 8
```

Why tuples?

- FASTER: iterating a tuple beats a list.
- SAFE: immutable; so data cannot change
- by accident.
- Good as fixed records. / dictionary keys.

Tuple vs List

Both are ordered sequences with indexing and slicing. The big difference is mutability.

Feature	List	Tuple
brackets	[ ]	( )
mutable?	YES	NO
speed	slower	faster
use for	same type	fixed record
as dict key	no	yes

*

Pick the right one

- Data will change / grow -> use a LIST.
- Data is fixed + must not change -> TUPLE.

- A tuple of coordinates (x,y) or an RGB
- colour (r,g,b) are classic tuple uses.

A tuple ~~can~~ canNOT be changed once made.

Introduction to Dicts

A dictionary is a MAPPING between a set of keys and a set of values. Each key : value pair is called an ITEM.

item = key : value

Creating a dictionary

Items go inside { } curly braces, key and value joined by a colon, items by commas.

```
>>> dict1 = {}                    #empty dict
>>> dict2 = dict()                #also empty
>>> dict3 = {'Mohan':95,'Ram':89,
             'Suhel':92,'Sangeeta':85}
>>>dict3['Ram']                   #value by key
89
```

Key rules

- Keys must be UNIQUE + IMMUTABLE (num, str, tuple). Values can repeat, any type.

Accessing Items

A list / tuple is indexed by POSITION. A dictionary is indexed by its ~~position~~KEY. Each key acts as the index and maps to one value. So the ORDER of items does not matter.

```
>>> dict3 = {'Mohan':95, 'Ram':89,
             'Suhel':92, 'Sangeeta':85}
>>> dict3['Ram']
89
>>> dict3['Sangeeta']
85
>>> dict3['Shyam']    #missing key
KeyError: 'Shyam'
```

Watch out

- Reading a missing key with [ ] raises
- KeyError.
- Use .get(key) to avoid the crash - it
- returns None if the key is absent.

'Ram' always maps to 89, wherever it sits.

Dicts are Mutable

Unlike a tuple, a dictionary CAN be changed after it is created - add, update or delete.

Add & modify

```
>>> d = {'Mohan':95, 'Ram':89}
>>> d['Meena'] = 78        #NEW key adds
>>> d['Ram'] = 90          #old key updates
>>> d
{'Mohan':95, 'Ram':90, 'Meena':78}
```

Delete : del / pop / popitem / clear

```
>>> del d['Ram']          #delete one item
>>> d.pop('Mohan')        #remove + return
95
>>> d.popitem()           #remove last item
>>> d.clear()             #empty the dict
>>> del d                 #remove the name
```

Membership & Traversal

Membership (in)

in / not in check for a KEY (not a value), returning True or False.

```
>>> d = {'Mohan':95, 'Ram':89, 'Suhel':92}
>>> 'Suhel' in d
True
>>> 'Suhel' not in d
False
```

Traversing a dictionary

A for loop visits every item. Two ways:

```
#Method 1 - loop over keys
>>> for key in d:
        print(key,':',d[key])

#Method 2 - loop over items()
>>> for k,v in d.items():
        print(k,':',v)
```

Dictionary Methods

keys / values / items

```
>>> d = {'Mohan':95, 'Ram':89}
>>> d.keys()
dict_keys(['Mohan', 'Ram'])
>>> d.values()
dict_values([95, 89])
>>> d.items()
dict_items([('Mohan',95),('Ram',89)])
```

get / update / setdefault

```
>>> d.get('Ram')          #safe read
89
>>> d.get('Sohan')        #missing -> None
>>> d.update({'Geeta':89})  #merge
>>> d.setdefault('Ann',0)   #add if new
0
```

Worked : Word Frequency

Program 10-7 : count how many times each character appears in a string, using a dict.

```
st = input('Enter a string: ')
dic = {}                    #empty dict
for ch in st:
    if ch in dic:           #seen before
        dic[ch] += 1
    else:                   #first time
        dic[ch] = 1
for key in dic:
    print(key,':',dic[key])
```

*

Output for 'HelloWorld'

```
H:1  e:1  l:3  o:2
W:1  r:1  d:1
```

The key is the character; the value is its running count. Same idea counts words in a sentence (split first, then count).

Worked : Phonebook

*

Store friends' names as keys and phone numbers as values, then look one up.

```
phone = {}
phone['Amina'] = 9812345670
phone['Joseph'] = 9900112233
name = input('Whose number? ')
if name in phone:
    print(phone[name])
else:
    print('Not found')
```

Marks lookup (Program 10-5 idea)

```
ODD = {1:'One',3:'Three',5:'Five',
       7:'Seven',9:'Nine'}
>>> len(ODD)        #5
>>> 7 in ODD        #True
>>> ODD.get(9)      #'Nine'
>>> del ODD[9]      #remove key 9
```

Appendix: Tuple Methods

Function	What it does
len(t)	number of elements
tuple(s)	build tuple from sequence
t.count(x)	times x appears
t.index(x)	index of first x
min(t)	smallest element
max(t)	largest element
sum(t)	total of elements
sorted(t)	new sorted LIST

Remember

- count() and index() are METHODS (t.count).
- len/min/max/sum/sorted are built-in FUNCS.
- sorted() returns a list, tuple stays same.
- index() on a missing value -> ValueError.*

No append / remove: tuples are immutable.

Appendix: Dict + Compare

Dictionary methods

d.keys()	list of keys
d.values()	list of values
d.items()	list of (k,v) tuples
d.get(k)	value or None
d.update(d2)	merge d2 into d
d.setdefault	add k if absent
d.pop(k)	remove k, return value
d.clear()	empty the dict

Tuple vs List vs Dict

	Tuple	List	Dict
marks	( )	[ ]	{ }
ordered	yes	yes	yes
mutable	no	yes	yes
index by	pos	pos	key
dupes	yes	yes	keys no

Recall & Mistakes

Common mistakes

(1) Single-elem tuple

(20) is an int; (20,) is a tuple.

(2) Changing a tuple

t[i]=x -> TypeError (immutable).

(3) Missing key

d['x'] -> KeyError; use d.get('x').

(4) Bad dict key

list can't be a key (mutable).

Quick recall

Tuple ( ) immutable ; Dict { } key:value

- Mnemonic 'CITMS' tuple funcs: Count,
- Index, Tuple, Min/Max, Sum/Sorted.

PYQ trend

- Find output: slicing, count/index (2 mk).
- Tuple vs list; dict methods keys/get (1-2mk).