* immutable - can't be changed. (fixed)
(lists are mutable, tuples are not)

Python - Tutorial 3

- Tuples: (homogeneous / heterogenious data)
(almost similar to list but its immutable)
c = ( ) , type(c) = tuple.

* almost all the operations of list are possible here too, like reversibility, extracting data etc. but replacement is not possible (this is called immutability).
So, tuples are used where we need fix data like days in a week.

ex:- c = (2, 3, 'Madhu', 3+4j).
c[0] → 2.
but, c[0] = 'Jain' → c ↷ error (tuples doesn't support assignment)
→ c.count(3) → 1 (as 3 appears only once).
→ c.index(3+4j) → 3 (as 3+4j is available at 3rd index).

[like, if we have any data multiple times & we use index operation so, it will give index of first position of that particular data.].

→ a = 4, 5, 6, 7 ⇒ type(a) = tuple
a → (4, 5, 6, 7) means

→ tuple doesn't supports append.

→ mapping:

a, b, c = 4, 6, 2.

a → 4
b → 6
c → 2

{ one to one mapping / assign multiple variables in single line. }

→ nested-tuples:

a = ((2,3,4),(5,2),2) ✓

→ tuple inside list:

a = [(4,2,3),(1,8),9]

a[1][1] → 8

→ list inside tuple:

a = ([1,2,3]) | type(a) → list

(if) b = ([2,4,5],[8,9]) | type(b) → tuple

→ So, even if you wish to manipulate tuple you can do it by first converting into list, then manipulate, then change back

a = (4,5,6,7).

c = list(a).

c[2] = 'MADHU'

c → [4, 5, 'MADHU', 7]

type(c) → (4, 5, 'MADHU', 7).

- **Sets :** (unordered collection of unique elem)

  s = { } , type(s) → dictionary

  s = {2,3}, type(s) → set

→ s = {2, 3, 22, 2, 3, 22, 'MADHU', 3}

  s → {2, 3, 22, 'MADHU'} (unique).

  (no duplicate data, only unique).

  x = set(), type(x) → set.

  list inside set :

  s = {[2,3], [4,5,6]} → error

  so, we can hold only primitive elements like, int:, float, etc.

→ unordered :

  x = {1, 2, 3, 1, 5, 1, 1, 2, 5}.

  x[0] → unsubscriptible (error)

```
for i in x:
    print(i)
```
→ 1, 2, 3, 5

→ set-functions :

- x.add(4) (: when we wish to add some new data.

  x → ~~{1, 2, 3, 1, 4, 5, 1, 1, 2, 5}~~ {1, 2, 3, 4, 5}

  (not like append that adds data at end)

→ set to list, list to set (possible).

• Dictionaries:

v = { }, type(v) → dictionary

- v = {'key': "MADHU"}
  v[0] → error
  v['key'] → 'MADHU'
- v = {'key': "MA", 'key1': 0, 'key2': 'abc',
  'key3': [list], 'key4': '[list]' }
  v['key3'] → error ✗(: list not allowed)
  v['key4'] → '[list]'(: '' means string)

→ duplicate key is overwritten:

v = {'key': 1, 'key1': 2, 'key': 3}

v → {'key': 3, 'key1': 2}.

→ v = {'key': 1, 'key': [6, 9, 8, 'MA']}

v {'key': [6, 9, 8, 'MA']}.

→ data as set, as tuple is also possible

→ dictionary inside dictionary:

d = {'a': 34, 'b': 345, 'd': {8: 45, 7: 87}}
d['d'][8] → 45

→ iteration:

for i in d:
  print i → a, b, d (only key, not data)

for i in d:
  print(i + " " + str(d[i])) → 34, 345, {8:45, 7:87}

(heterogeneous data) typecast (because all data is not in one)

key allowance (int, string, boolean (True/False))

→ d.keys() → dict_keys(['a', 'b', 'd'])
→ d.values() → dict_values([34, 345, {8:45, 7:87}])
→ Insert new key & value:
d['key45'] = "MADHU"
d → {'a':34, 'b':345, 'd':{8:45, 7:87}, 'key45':'MADHU'}.
update value of any key:
d['b'] = [8, 10, 12]
d → {'a':34, 'b':[8,10,12] 'd':{8:45, 7:87}, 'key45':'MADHU'}

Note: { If any key is not already available then new insertion is done otherwise updation. }.

→ Nesting with dictionaries:
d = {'key1':{'nestkey':{'subnestkey':786}}}
d['key1']['nestkey'] → {'subnestkey':786}
→ tuples of items:
dict = {'key1':1, 'key2':2, 'key3':3}
dict.items → dict_items([('key1',1), ('key2',2), ('key3',3)])
→ Dictionary comprehension:
eg: l = ['MADHU', 'NIDHI', 'TAPUR', 'SHIMU']
{i[0]:i for i in l}
↳ {'M':'MADHU', 'N':'NIDHI', 'T':'TAPUR', 'S':'SHIMU'}

(wherever you don't want to use body of any function say if/else/for... or even user-defined then use keyword pass).

Eg: squares of even numbers:

```
{x:x**2 for x in range(10) if x%2==0}
{0:0, 2:4, 4:16, 6:36, 8:64}
```

- Functions: (user-defined).

```
def name-of-function(arg1, arg2):
```

```
def test():          (: without argument).
    print("FIRST FUCTION")
```

test() → FIRST FUNCTION

type(test()) → NoneType

```
def test():
    print("Hello")   (: print always gives you NoneType)
a = test()
```

a → Hello (type(a) = NoneType)

a+"MADHU" → error (NoneType + str) X

```
def test()
    return "Hi"   (return any type int, float, list, tuple, dic, etc)
a = test()
```

a → 'Hi' (type(a) = str)

a+" kumar" → 'Hi kumar'

multiple value return:

```
def test():
    return 5+8j, 3, 6, 8
b = test()
```

b → (5+8j), 3, 6, 6)

```
(u,v,w,x) = test()
v → 3
b,_,_,_ = test()
_ → 8   (over-right).
```

→ function with argument:

```
def test(a,b)   (arg of some type).
    return a+b
test([2,8,4,4],[5,6,6])
[2,8,4,4,5,6,6]
```