By Sai Kumar

PYTHON IDENTIFIERS – PART B

Tuple Data Type

tuple data type is exactly same as list data type except that it is immutable. i.e., we cannot change values.

Tuple elements can be represented within parenthesis.

Eg:

1) **t=(10,20,30,40)**

2) **type(t)**

<class 'tuple'>

3) **t[0]=100**

TypeError: 'tuple' object does not support item assignment

4) >>> **t.append("sai")**

AttributeError: 'tuple' object has no attribute 'append'

5) >>> **t.remove(10)**

AttributeError: 'tuple' object has no attribute 'remove'

Note: tuple is the read only version of list

Range Data Type:

range Data Type represents a sequence of numbers.

The elements present in range Data type are not modifiable. i.e., range Data type is immutable.

Form-1: **range(10)**

generate numbers from 0 to 9

Eg:

r = range(10)

for i in r : print(i) → 0 to 9

Form-2: **range(10, 20)**

generate numbers from 10 to 19

Eg:

r = range(10,20)

for i in r : print(i) →10 to 19

Form-3: **range(10, 20, 2)**

2 means increment value

Eg:

r = range(10,20,2)

for i in r : print(i) → 10,12,14,16,18

We can access elements present in the range Data Type by using index.

Eg:

r = range(10,20)

r[0] → 10

r[15] → IndexError: range object index out of range

We cannot modify the values of range data type

Eg:

r[0] = 100

TypeError: 'range' object does not support item assignment

We can create a list of values with range data type

Eg:

1) >>> **l = list(range(10))**

2) >>> **l**

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

set Data Type:

- If we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.

1) Insertion order is not preserved

2) Duplicates are not allowed

3) Heterogeneous objects are allowed

4) Index concept is not applicable

5) It is mutable collection

6) Growable in nature

Eg:

1) **s={100,0,10,200,10,'sai'}**

2) **s # {0, 100, 'sai', 200, 10}**

3) **s[0]** → TypeError: 'set' object does not support indexing

- set is growable in nature, based on our requirement we can increase or decrease the size.

1) >>> **s.add(60)**

2) >>> **s**

{0, 100, 'sai', 200, 10, 60}

3) >>> **s.remove(100)**

4) >>> **s**

{0, 'sai', 200, 10, 60}

frozenset Data Type:

- It is exactly same as set except that it is immutable.

- Hence, we cannot use add or remove functions.

1) >>> **s={10,20,30,40}**

2) >>> **fs=frozenset(s)**

3) >>> **type(fs)**

<class 'frozenset'>

4) >>> **fs**

frozenset({40, 10, 20, 30})

7) >>> **for i in fs:print(i)**

... [Enter]

40

10

20

30

8) >>> **fs.add(70)**

AttributeError: 'frozenset' object has no attribute 'add'

16) >>> **fs.remove(10)**

AttributeError: 'frozenset' object has no attribute 'remove'

dict Data Type:

- If we want to represent a group of values as key-value pairs then we should go for dict data type.
- Eg: d = {101:'sai',102:'ravi',103:'shiva'}
- Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.

Eg:

1) >>> **d={101:'sai',102:'ravi',103:'shiva'}**

2) >>> **d[101]='sunny'**

3)>>> **d**

{101: 'sunny', 102: 'ravi', 103: 'shiva'}

We can create empty dictionary as follows

4) **d={ }**

We can add key-value pairs as follows

5) **d['a']='apple'**

6) **d['b']='banana'**

7) **print(d)**

Note: dict is mutable and the order won't be preserved.

Note:

1) In general, we can use bytes and bytearray data types to represent binary information like images, video files etc.

2) In Python2 long data type is available. But in Python3 it is not available and we can represent long values also by using int type only.

3) In Python there is no char data type. Hence, we can represent char values also by using str type.

Summary of Datatypes in Python 3

Datatype	Description	Is Immutable?	Example

Int	We can use to represent the whole/integral numbers	Immutable	>>> a=10
			>>> type(a)
			<class 'int'>

Float	We can use to represent the decimal/floating point numbers	Immutable	>>> b=10.5
			>>> type(b)
			<class 'float'>

Complex	We can use to represent the complex numbers	Immutable	>>> c=10+5j
			>>> type(c)
			<class 'complex'>
			>>> c.real
			10.0
			>>> c.imag
			5.0

Bool	We can use to represent the logical values (Only allowed values are True and False)	Immutable	>>> flag=True
			>>> flag=False
			>>> type(flag)
			<class 'bool'>

Str	To represent sequence of Characters	Immutable	>>> s='durga'
			>>> type(s)
			<class 'str'>
			>>> s="durga"
			>>> s='''Durga Software Solutions... Ameerpet'''
			>>> type(s)
			<class 'str'>

bytes	To represent a sequence of byte values from 0-255	Immutable	>>> list=[1,2,3,4]
			>>> b=bytes(list)
			>>> type(b)
			<class 'bytes'>

bytearray	To represent a sequence of byte values from 0-255	Mutable	>>> list=[10,20,30]
			>>> ba=bytearray(list)
			>>> type(ba)
			<class 'bytearray'>

range	To represent a range of values	Immutable	>>> r=range(10)
			>>> r1=range(0,10)
			>>> r2=range(0,10,2)

list	To represent an ordered collection of objects	Mutable	>>> l=[10,11,12,13,14,15]
			>>> type(l)
			<class 'list'>

tuple	To represent an ordered collections of objects	Immutable	>>> t=(1,2,3,4,5)
			>>> type(t)
			<class 'tuple'>

set	To represent an unordered collection of unique objects	Mutable	>>> s={1,2,3,4,5,6}
			>>> type(s)
			<class 'set'>

frozenset	To represent an unordered collection of unique objects	Immutable	>>> s={11,2,3,'Durga',100,'Ramu'}
			>>> fs=frozenset(s)
			>>> type(fs)
			<class 'frozenset'>

dict	To represent a group of key value pairs	Mutable	>>> d = {101:'durga', 102:'ramu', 103:'hari'} >>> type(d) <class 'dict'>