26/10/2020

Sets

→ A set is a collection of some of the homogenous and heterogenous non repetitive data items that are enclosed b/w the pair of flower braces and separated by using comma operator.

→ A set is a unordered collection where the elements will not be in the order how it should be.

→ As the set is a random collection (the place where we insert might not be same as the place where it is present because of that we can say indexing is not possible

→ Set will not allow any of the mutable types in it.

[list, set & dictionary]

Syntax → Var = { Val1, Val2 ——— Valn }

should not have duplicates, if it is there it will be eliminated

Example

(1) a = {5, 4, 3, 2, 1}

→ a

type (a) → set

(2) b = {0.5, 1, 1.5, 2.5, 3, 3.5}

(3) c = {1, 2, 3, 'a', 'b', 'c'}

(4) d = {1, 2, 1, 2, 1, 3}

(5) e = {1, 2, 3, 3+1j, (1, 2, 3)}

(6) d_1 = {1, 2, 3, 3+1j, (1, 2, 3), [1, 2]}

→ Error

(7) d_2 = {1, 2, 3, 3+1j, (1, 2, 3), [1, 2], {a:1, b

→ Error

Set is a mutable type (ie we can modify using some built in function but can't access using position (or) Subscripting)

(ie) You can't able to modify if using index but you can by using set of inbuilt (or) built in functions

Dictionary

→ Collection of the data in the form of key value pairs

→ key will describe what the values are

Key value pairs are separated by using Comma operator and are enclosed b/w the pairs of flower Braces and keys & values are separated by using Colon operator.

→ keys Cannot be list, Set & dictionary but the preferred key type is String

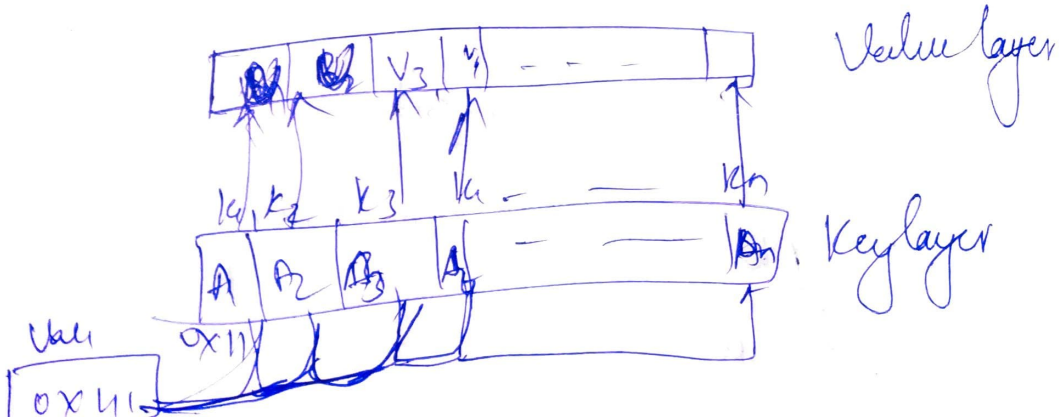
→ Values can be of any type

Syntax

Var = { k_1 : v_1 , k_2 : v_2 , k_3 : v_3 ..... k_n : v_n }

↓ ↓

Keys Values

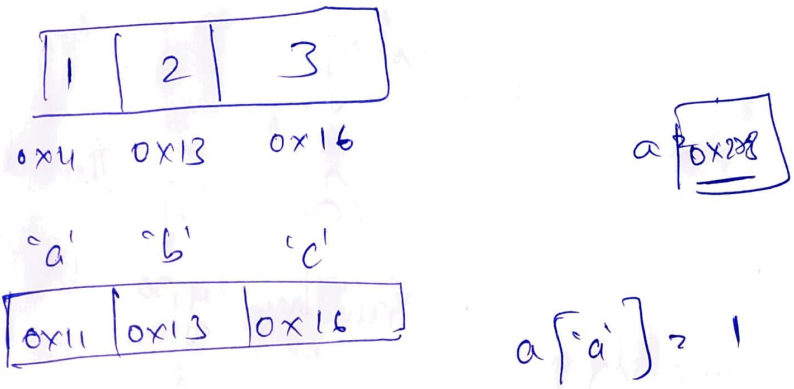


→ To access value layer we have to go through key layer as only key layer is visible to outside.

→ To get value that is associated with the key

Var[key] ⇒ gives value associated with the key

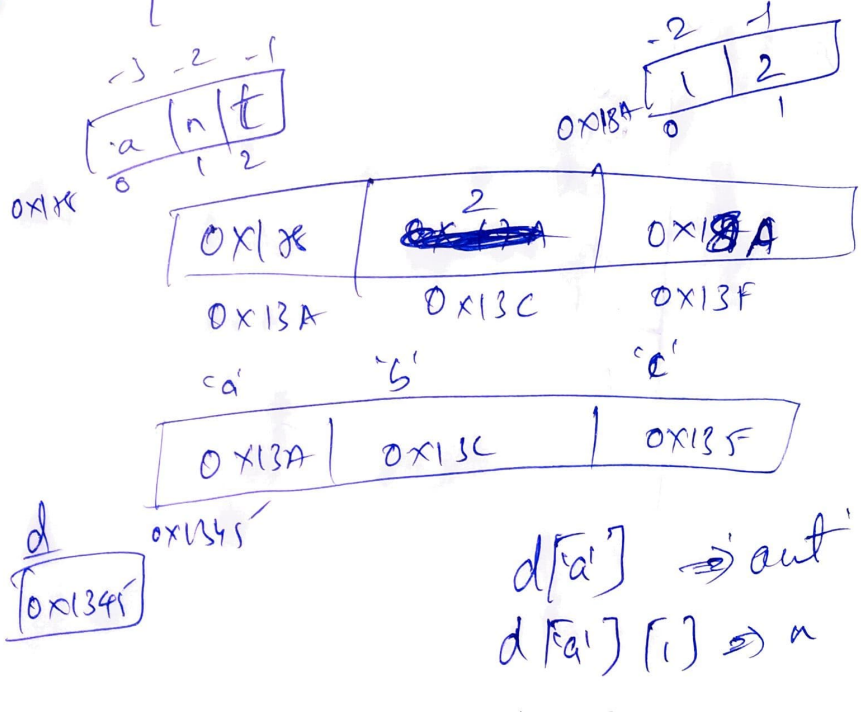
→ a = {'a': 1, 'b': 2, 'c': 3}



a['a'] = 1

a['b'] = 2

→ d = {'a': 'aut', 'b': 2, 'c': [1,2]}



d['a'] ⇒ 'aut'

d['a'][1] ⇒ n

a = {'a': 'apple', 'b': 'bat', 'c': 'cat'}

a = {'a': 'apple', 'b': 'bat', 'c': 'cat', 'b': 'ball'}

a['a'] = "america"

a['e'] = Egg

len(a) type(a)

27/10/2020

Type Casting

Process / phenomenon of converting the data from one type to another type is called as type casting.

Type casting is done using a syntax

Var = datatype (Var / value)

- int
- float
- str
- list
- Tuple
- set
- dict

a = {'a': 'apple', 'b': 'bat', 'c': 'cat'}

a = {'a': 'apple', 'b': 'bat', 'c': 'cat', 'b': 'ball'}

a['a'] = "ameeza"

a['e'] = Egg

len(a) type(a)

27/10/2020

Type Casting

Process / phenomenon of Converting the data from one type to another type is called as type Casting.

Type Casting is done using a syntax

Var = datatype (Var / Value)

- int
- float
- Str
- list
- Tuple
- set
- dict

Source type	destination type	Comments
int	float, Complex, str	all simple value type can be converted only collection type
float	int, str, Complex	decimal part will be removed [truncated] causing data loss
Complex	str	As Complex is having 2 set of values we cannot able to convert it to other type
str	int, float	only if str contains only num with or without .
	Complex	only if it is in form a+bj
	set, list, tupple	in set order of elements change, duplicates will be eliminated
list	tupple, set, dict, str (but not recommend)	Data loss will happen when we convert to set
		When we have 2 data items in a element we can ie list inside a list or list inside tupple
Tupple	list, set, str	str (operation)
	dict	set → order will become undecided & duplicates will be removed
		data should be in form of nested collection
set	list, tupple, str	Unordered will be
	dict (key value pair)	Order → list mutable → tupple → immutable
dict	list, tupple, set	#values lost
	str	str to dict not possible

int

Eg >>> a=10
→ 10.0 , type(a)
<class 'int'>
>>> a=float(a)
→ 10.0 , type(a)
<class 'float'>

a = complex(a)
→ 10+0j
a = complex(10,12)
→ 10+12j
c = str(15)
c[0] , c[1]
c → '15'

float

a=12.99
int(a) ⟹ 12
complex(15)
⟹ (15+0j)
c=str(12.75)
c='12.75'

complex(1.5,2)
→ 1.5+2j
complex(imag=2.4)
2.4j
type(c) =>
<class 'str'>

list

a=[1,2,3]
⟹ a ⟹ [1,2,3]

str(a)
tuple(a)
set(a)
dict(a)

tuple

a=(1,2,3)
list(a)
b=[1,2,3,4,5,6,1,2,7]
set(b) set(b)
tuple(set(b))
c=[(1,'a'), [2,'4'], {'c', 3}]
list(c) ⟹

set

s = {1, 2, 'b', 'c', 8}

b=list(a)
tuple(a)
str(a)
set(str(a))
a=((1,2),(3,4))
dict(a)

str

int("312") error
int("a312") error
int("312")
→ 312
float("312.54")
⟹ 312.54

complex("3+1j")
(3+1j)
s="hello world"
list(s)
tuple(s)
set(s)

dict

a={'a':1, 'b':2, 'c':3}

list(a) set(a)
tuple(a) str(a)

but vice versa not possible

Default & Non default values in a data type

In python each and every data type we have will be having two types of data in it

1. default values
2. non default values

→ Type of values which are equivalent to the Boolean value called 'False'

non default values → All the set of values that are equivalent to the boolean value 'True'

default values

Datatype	default value	
int	0, int()	→ int() gives 0
float	0.0, float()	" 0.0
Complex	0j, Complex()	" 0j
Bool	False, bool()	" False
str	'', "", """ """ or str()	
list	[], list().	
tupple	(), tupple()	
dict	dict(), {}	
Set	Set()	

Any default values converted to other type gives false

Eg bool(0) => false bool(dict())
bool({}) => false
bool(float()) => false
bool([]) => false

If we wanted to create a variable of any type with no value then we use syntax

var = datatype()

---

other than default value whatever the value that we have those are called as non-default values and all the non-default values are equivalent to 'True'

Eg bool(1.5), bool('j'), bool(0.1) } => True

String

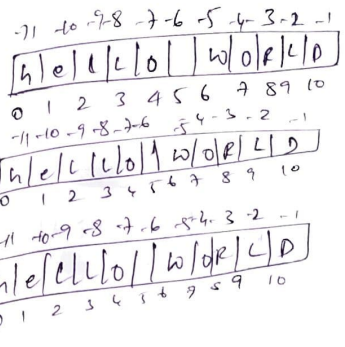
phenomenon of breaking the collection into pieces of the specified size at the specified position

Accessing the sequence of elements b/w limits

Str: a = "hello world"

list: a = ['h','e','l','l','o',' ','w','o','r','l','d']

Tuple: a = ('h','e','l','l','o',' ','w','o','r','l','d')



Syntax

Var[startindex : Endindex+1]

Startindex < endindex

If startindex is zero eliminate it then use Var[: endindex+1]

If Endindex+1 is equal to length the eliminate it & use Var[startindex :]

Either +ve positive index or negative index internally everything will get converted to +ve index only.

i.e +ve = length + (-ve)

identifying an element using slicing

Var[index]

a = 'hello'

a => 'hello'

a[2] →

a[3] →

a[-2] →

a[-4] →

b = 'hey all'

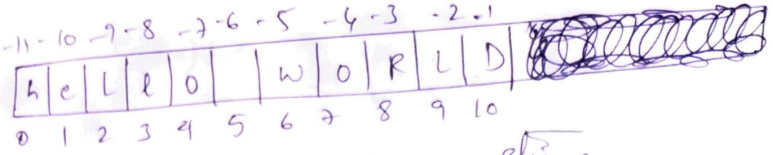
b[3] →

b[2] →

b[-6] →

b[-3] →

c = 'hello world'



Identify a substring from string

Var[0:7] => Var[si : ei+1]

a[2:6]

a[2:7]

a[-9:-4]

a[2:-4]

a[0:5] => 'hello'

a[-11:-6] => 'hello'

a[6:99] => 'world'

a[6:11] => 'world'

a[6:10] => 'world'

Var[si : ei+1]

↳ si → 0 (default)

↳ ei+1 → len(var) (default)

↳ a[:8] => 'hello wo'

a[4:] => 'o world'

a[8] = 'x'

a[4] = '0'

Var[si : ei+1 : updation]

→ a[0:11:2] => 'hlowrld'

→ a[2:99:3] => 'l x'

↑ fetching the subsequence from the collection of values with custom updation

updation → by default → 1

a[::] => 'hello world'

a => 'hello world'

a[0:2:0] → step cannot be zero

z = [1,2,3,[44,55,66,77],99]

z[2] = 3

z[3] => [45, 55, 66, 77]

z[3][1:] => 55

z[3][1] → [55, 66, 77]

c = [1, 2, 44, 55, 'hello', 'world']

c[-2][2:4]

→ 'll'

Case-1

updation → +ve

→ si < ei+1

updatin ie -ve

→ si > Ei+1

↓ len(var)-1 (default)

Case-2

↓ len(var)+1 (default)

If it not we get empty string

↳ Eg a[8:4:-1]

Updation can be +ve (or) -ve

when updation +ve case-1

& when updation -ve case-2

a[8:4:-1]	a[4:-12]
'row'	::
a[4:-12:-1]	a → 'hello world'
'olleh'	

z = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

z

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

z[1:12] → [2, 4, 6, 8, 10]

z[::-2]

z[::-1]

x = 'hello world'

x[::-1]

x = x[::-1]

Inbuilt functions on tuple

No allowance to do modifications in tuple

we do not have much inbuilt functions.

→ hence tuple is immutable : only few inbuilt function and no allowance of insertion & deletion and the only inbuilt functions on tuple are.

① count()

Var.count(Element)

function which counts the occurence of Element specified.

a = (1, 3, 2, 30, 40, 598, 20, 3, 2, 1, 3, 4)

a.count(3) → 3 (Homogenous Elements)

a.count(99) → 0

a = (1, 3.5, 3+4j, True, 'hai', [1, 2, 3, 4], 1, 45.66, 23, 'hai')

a.count('hai') → 2 (Heterogenous Elements)

a.count(1) → 1

(2) index()

Var.index(element)

Var.index(Element, startindex)

Var.index(Element, startindex, Endindex+1)

Inbuilt functions on Set

As set stores the value in random manner we dont have the inbuilt functions related to indexing

→ add()

Var.add(Element)

It is an inbuilt function used to add an element to a set Collection

the Element should be single value type, string, tupple

a = {1, 2, 3, 4}

a.add(69)

a ⇒ {1, 2, 3, 4, 69}

---

a.add('hai')

a ⇒ {1, 'hai', 2, 3, 4, 69}

→ remove()

Var.remove(Element)

It is a function used to remove the specified Element from Set Collection.

a.remove(69)

a ⇒ {'hai', 1, 2, 3, 4}

a.remove(2)

a ⇒ {1, 'hai', 3, 4}

→ discard()

Var.discard(Element)

a.discard(99)

⇒ no message / NOS.

a.discard('hai')

a ⇒ {1, 3, 4}

Use to remove a specified Element if the Element is not present it will not throws any error.

② index()

Var.index(Element)

Var.index(Element, startindex)

Var.index(Element, startindex, Endindex+1)

Inbuilt functions on Set

As set stores the value in random manner we dont have the inbuilt functions related to indexing

→ add()

Var.add(Element)

It is an inbuilt function used to add an element to a set collection

the element should be single value type, string, tupple.

a = {1, 2, 3, 4}

a.add(69)

a ⇒ {1, 2, 3, 4, 69}

a.add('hai')

a ⇒ {1, 'hai', 2, 3, 4, 69}

→ remove()

Var.remove(Element)

It is a function used to remove the specified element from set collection.

a.remove(69)

a ⇒ {'hai', 1, 2, 3, 4}

a.remove(2)

a ⇒ {1, 'hai', 3, 4}

→ discard()

Var.discard(Element)

a.discard(99)

⇒ no message / NOR.

a.discard('hai')

a ⇒ {1, 3, 4}

use to remove a specified element if the element is not present it will not throw any error.

→ pop()

Var.pop() Removes & Return random value from set & remove

pop from empty set is Error.

→ intersection

Var1. intersection (Var2)

Var1. intersection (Var2, Var3, ---- Varn)

It is an inbuilt function which is used to Return the Common Element from the specified Sets.

→ intersection_update

Var1. intersection_update(Var2)

Var1. intersection_update (Var2, Var3 - - - Varn)

Eg a = {1, 2, 3, 5}

b = {5, 3, 7, 8}

a. intersection_update (b)

⇒ a = {3, 5}

Use to perform the intersection operation and update the actual result with the existing set

① Clear()

Var. clear()

Clear all the values of the set.

② union()

Var1. union(Var2)

Var1. union(Var2, Var3 - - - Varn)

union is a function which works like a combination for set.

③ difference()

Var1. difference(Var2) (Var1 not to Var2 Var1 - Var2)

Var2. difference(Var1) (Var2 - Var1)

Var1. difference(Var2 - - - Varn)

↓

It is a function used to find a difference of one set collection values with the another set collection values

(16) difference_update()

Var1.difference_update(Var2)

Var1.difference_update(Var2, Va---)

Inbuilt functions on dictionary

(1) get() inbuilt function which is use to return the corresponding value of the specified key

Var.get(key)

z = {'a':1, 'b':2, 'c':3}

Var.get('c') ⇒ z['c']

⇒ 30

(2) popitem()

Var.popitem()

inbuilt function used to remove the last key & value pair from the dictionary

zz.popitem()

(3) pop() Var.pop(Element) → Specify keyname

It removes the specified key & value pair from dictionary.

(4) update()

Var.Update(Var2/dictionary)

Eg z = {a:1, b:2}

c = {c:3, e:5}

z.update(c)

z = {'a':1, 'b':2, 'c':3, 'e':5}

It is an inbuilt function use to update the existing dictionary (specified dictionary)

(5) clear()

Var.clear()

(6) keys() Var.keys()

(7) values Var.values()

String

L → R → +ve index (Recommended) Endindex-1

R → → -ve index (Recommended) Endindex-1

---

operators

operators are the set of special symbols that are been assigned with a pre-defined task or operation

1. Arithmetic operator
2. Logical operator
3. Relational operator
4. Bitwise operator
5. Assignment operator
6. Membership operator
7. Identity operator

Arithmetic operator

'+', '-', *, /, //, ** Etc

(1) (+) (addition / Concatenation)

Var = Numeric + Numeric

Eg 3+5 = 8

5.2 + 3.6

8.8

3.9+4 = 7.9

(3+5j) + (4+5j)

=

3+5j + 4

7+5j

Collection + Collection

Same type (Homogeneous type)

ie Str + Str / tuple + tuple

List + List

but not allowed with sets & dict

Var = [Val1, Val2, Val3] +

[Val11, Val22]

= [Val1, Val2, Val3, Val11, Val22]

Eg 'a'+'b', 'ab' , [1,2] + [3,4], [1,2,3,4]

(2) Subtraction

Var = Numeric - Numeric

Subtraction will be used only on simple type value such as Numeric → int, float, Complex.

3-2 = 1

3-1.5 = 1.5

4-3.2 => 0.8

(5+2j) - (4-3j)

-1+4j

③ Multiplication / Replication

Var = Numeric * Numeric

3 * 2
= 6

1.5 * 3
= 4.5

2j * 1.5
= 3j

(3+2j) * (3-2j)
9 + 6j - 6j + 4
13

Collection * integer.

integer * Collection.

3 * 'a' = 'aaa'

3 * 'ab' = 'ababab'

3 * [1,2,3]
[1,2,3,1,2,3,1,2,3]

Set & dict cannot be used

④ True division (/)

P.S we get the Exact value in the division operation we call it as True division

Var = Numeric / Numeric

= 3/2 = 1.5

=> 3/3 => 1

=> $\frac{(1+2j)}{(3+2j)} + \frac{(3-2j)}{(3-2j)}$

$\frac{3+6j-2j+(-4)}{13} \quad \frac{2+4j}{13}$

⑤ floor division

int — real value

decimal (float)

int — floor value

[2, 1.5, 1] — value

Var = Numeric // Numeric

5//2 = [5/2] = [2.5] = 2

3//1.5 = [3/1.5] = [2] = 2