Ex: a, b, c, d, e = 10, 10, 20, 30, 20

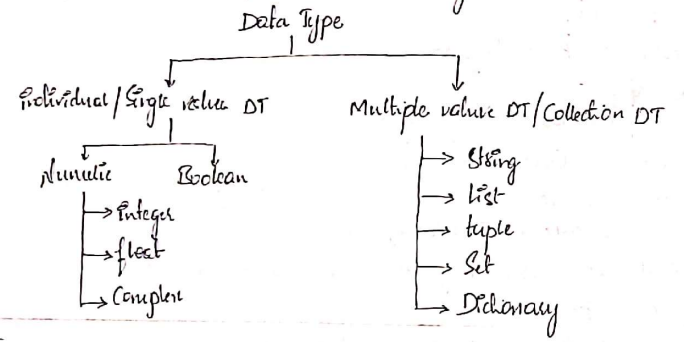
```
>>
>> f = 789
   d = 10
   e = 30
```

Var Space	Var Space
a	0x11 / 10
[0x11]	
b	
[0x11]	

Data Types:- Data type is something, which will define the type of the value & size of the value. Stored into a variable.

> Based on the size of the value data type got classified into 2 categories,

1. Single value datatype: It is a data type, where we are going to store a single value into a single variable.
2. Multi value datatype / Collection DT :- It is a datatype, where we are going to store multiple value into a single variable.



Integer :- > It is real number without decimal point. (-∞, ... -1, 0, +1 .... +∞)

> Integer can be either +ve or -ve

> Each & every datatype will be having 2 types of values

1. Default value
2. Non-default value

1. **Default value :-** It is an initial value, which will be internally considered as False.
2. **Non-default value :** Apart from default value, all the values are considered as non-default values, which will be internally equal to True.

> 0 is the default value for Integer Number.

> **type ()**, It is a function which is used to check the type of the value stored into a variable.

Syntax: `type (var/value)`

> **bool ()**, It is a function which is used to check whether the value is internally equal to True / False.

Syntax: `bool (var/value)`

Ex: a = -58, n = 78

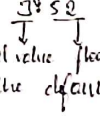
Var Space	value Space
a	0x12 / -58
[0x12]	
n	0x11 / 78
[0x11]	

```
>>> n=78
>>> type(n)
<class 'int'>
>>>
>>> a = -789
>>> type(a)
<class 'int'>
>>>
>>> b=0
>>> type(b)
<class 'int'>
>>>
>>> bool(0)
False
>>> bool(56)
True
>>> bool(-6)
True
```

```
>>> bool(012)
SyntaxError: invalid token
```

2) <float> :- > It is real number with decimal point (ex: -1.2, 0, 1.2 etc)

> float can be either +ve / -ve.



> 0.0 is the default value for float number.

> Ex: m = 3.7

var Space	val Space
m (0x14)	0x14 3.7

```
>>> a = 8.9
>>> type(a)
<class 'float'>
>>> c = -9.87
>>> type(c)
<class 'float'>
>>> type(0.0)
<class 'float'>
>>> bool(0.0)
False
>>> bool(a)
True
>>> 9.
9.0
>>> .6
0.6
>>> .
SyntaxError: Invalid Syntax
```

3) Complex: Complex is a number, which is combination of both Real & Imaginary parts.

- the number which is in the form $a \pm bj$ is called as Complex number.

$a \pm b\,j$ → Imaginary number ($j = \sqrt{-1}$)

a (Real part), b (Imaginary part)

> It is not possible to use any other character for Imaginary number except J or j.
> If we use 'J', internally it will get converted into 'j'.
> We cannot write an independent j without value for 'b'.
> It is possible to alter the position of Real & Imaginary part but not b and j.
> 0j is the default value for Complex number.

Ex: a = 5+6j

var Space	val Space
a (0x11)	0x11 5+6j

Ex:

```
>>> a = 6+4j
>>> type(a)
<class 'complex'>
>>>
>>> 5-4j
(5-4j)
>>> 6+7i
SyntaxError: Invalid Syntax
>>> 7+3K
SyntaxError: Invalid Syntax
>>> 6+7J
(6+7j)
>>> b=8j
>>> type(b)
<class 'complex'>
>>> b
8j
>>> b=67
>>> type(b)
<class 'int'>
>>> 7+j6
Traceback (most recent call last):
File "<pyshell #27>", line 1, in <module>
    7+j6
>>> 6+0j
(6+0j)
>>> 6+1j
(6+1j)
>>> a = 8+9.6j
>>> type(a)
<class 'complex'>
>>> bool(a)
True
>>> bool(0j)
False.
>>> bool(-12-6+j)
True
>>> bool(5+0j)
True
```

4) Boolean (bool): Boolean is a datatype which consist of only two values.

1. True
2. False

> True is internally considered as integer 1 & False is internally considered as integer 0.

> Since it is having only two values, True is considered as non-default value & False is considered as default value.

> Boolean datatype can be used in 2 scenarios

* As a value while creating variable

a = True
b = False

var space	val space
a	0x11
0x11	True
b	0x22
0x22	False

* As a resultant, while checking condition

a=10	a<b	a>b
b=20	True	False

> Boolean values are keywords.

Ex:-
```
>>> bool(a)
True
>>> a+b
1
>>> a+a
2
>>> a=10
>>> b=56
>>> a>b
True
>>> b>a
False
```

```
>>> a = True
>>> b = False
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
>>> bool(b)
False
>>> bool(a)
True
```

note:
> In a single block of memory, we can store a single data item, where the data item can be a value of an address.

5) String (str):-

> String is collection of characters enclosed between pair of single quote or pair of double quote or triple quote (""") & pair of single quote.

> The syntax to create string is

var = 'val1val2.... valn'

var = "val1val2.... valn"

var = """val1val2.... valn"""

> Where values can be either upper case or lower case or numbers or special symbol or combination of all.

> If we create a string in double quote or three pairs of single quote, internally it will take in the form of single quote.

> When we have single quote already inside a string then we should use double quote.

> The string which is created inside three pairs of single quote is called as doc string, which is used as documentation & it act as comment.

> If we start a string with single quote then we should end it with single quote & same for other two.

> '' is the default value for string datatype.

> In string there is no separation b/w the values.

> len() is used to find the no of characters present inside string.

Ex:
```
>>> st = 'good afternoon'
>>> type(st)
<class 'str'>
>>> st
'good afternoon'
>>> 'hai'
'hai'
>>> "hai"
'hai'
>>> '''hai'''
'hai'
```

```
>>> 'happy father's day'
Syntax Error: Invalid syntax
>>> "happy father's day"
"happy father's day"
>>> a = 'hai hello'
>>> len(a)
9
>>> "hello"
Syntax Error: EOL while scanning string literal
>>> a = ''
>>> bool(a)
False
>>> a = ' '
>>> bool(a)
True
```

Memory allocation for collection datatype:-

> As soon as compiler got to know that the value is of collection, it will create a layer of memory in value space.

> address will be given to the layer & that will get stored with respect to variable name in variable space.

> The entire layer will get divided into no of blocks which is exactly equal to length of collection.

Ex:- st = "ee Sala cup namde"

> If it is required to get the complete collection then we can use variable name.

> To get an individual values from the collection, we should make use of indexing

Indexing: → Indexing is a phenomenon of providing a sub-address to each & every values present inside the collection.

→ There are 2 types of indexing.

1. +ve Indexing
2. -ve Indexing

+ve Indexing: When we traverse from left to right of the collection then we can use +ve indexing, it will starts from 0 till length of collection -1 (len(col)-1)

-ve Indexing: When we traverse from right to left of collection then we should use -ve indexing, it will starts from -1 till -len(col).

Note:
→ If we use convinient there are 2 types of indexing, but Control will understand only the indexing.

→ If we give -ve index internally it will get converted into +ve index by using a formula +ve = len(col) + (-ve index)

St = 'good afternoon' = 14 + (-5)
= 14 - 5
= 09

val space	val space
St	0x21
0x21	

L←R

-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
g	o	o	d		a	f	t	e	r	n	o	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13

L→R

```
>>> St[-10]
'g'
>>> St[7]
't'
>>> St[-7]
't'
>>> St[-11]
'd'
```

> To access an individual value from the collection, we can make use of a syntax `var[index]`

> To modify the value based on indexing, we can use a syntax `var[index] = new value`

Note: Collection datatype got classified into 2 types.

1. Mutable
2. Immutable

1) **Mutable Collection:** It is a type of collection, where it is possible to modify the original value or adding a new value or removing an existing value.

2) **Immutable Collection:** It is a type of collection, where it is not possible to add a new value or remove an existing value or modify the original value.

> Since String will not allow the user to modify its original value, we can call it as an Immutable Collection.

Ex: s = 'python is very very very easyyyy'

val space	val space
s	0x11
0x11	

P	y	t	h	o	n		i	s		v	e	r	y		v	e	r	y		v	e	r	y		e	a	s	y	y	y	y
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

2. List (list) :-

Homogeneous Collection: It is a collection of data of same type.

Ex: 10, 20, 30, 40, 50,
int int int

Heterogeneous Collection: It is a collection of data of different type

Ex: 10, 2.3, 20, 3+4j
int float complex

> List is a collection of homogeneous data items & Heterogeneous data items enclosed b/w the pair of Square Braces.

> The Syntax to create list is `var = [val1, val2, val3.. valn]`

where the values are separated by comma (,) operator

> [ ], is the default value for list collection.

```
>>> a = [10,20,30,40]
>>> type(a)
<class 'list'>
>>> b = [1,2.3,5+6,7+8j,'hai']
>>> type(b)
<class 'list'>
>>> len(a)
4
>>> len(b)
5
>>> id(a)
2254271557336
>>> id(b)
2254266679752
```

```
>>> a = []
>>> bool(a)
False
>>> bool(b)
True
>>> c = [10]
>>> bool(c)
True
>>> type(c)
<class 'list'>
```

Ex: l = [10, 20, 30, 40, 50]

var space	val space
l	0x51
[0x51]	-5 -4 -3 -2 -1 | 10 | 20 | 30 | 40 | 50 | 0 1 2 3 4

l[0] = 100

> Since list will allow the user to modify its original values, we can call it as 'mutable collection'

> To modify the existing value, we can make use of a syntax

`var(index) = new value`

> To add a new value to the collection we can make use of following functions

1) `append()` :- It is a function, which is used to add a new value to the last position & syntax used is `var.append(value)`

2) `insert()` :- It is a function, which is use to add a new value to the required position & syntax is `var.insert(position, value)`

> To remove an existing value from the collection, we can use `pop()`

> pop will remove the last value from collection by default & the syntax used is `var.pop()`

> It is possible to remove the value based on specified index by using pop & the syntax used is `var.pop(index)`

Ex :- l = [10, 20, 30, 40, 50]

```
>>> l[0] = 100
>>> l
[100, 20, 30, 40, 50]
>>> l.append(200)
>>> l
[100, 20, 30, 40, 50, 200]
>>> l.insert(1, 1000)
>>> l
[100, 1000, 20, 30, 40, 50, 200]
```

```
>>> l.pop()
200
>>> l
[100, 1000, 20, 30, 40, 50]
>>> l.pop(0)
100
>>> l
[1000, 20, 30, 40, 50]
```

Memory allocation for Heterogenous list Collection:

Ex: a = [10, 2.8, [1,2,3], 'hello', 3+2j]

var space	val space
a	0x88
[0x88]	-5 -4 -3 -2 -1 | 10 | 2.8 | 0x11 | 0x22 | 3+2j | 0 1 2 3 4

0x11: -3 -2 -1 | 1 | 2 | 3 | 0 1 2

0x22: -5 -4 -3 -2 -1 | 4 | 0 | 1 | 1 | 0 | 0 1 2 3 4

Ex:

```
>>> a = [10, 2.9, [1,2,3], 'hello', 3+2j]
>>> a[2]
[1,2,3]
>>> a[3]
'hello'
>>> a[2][1]
2
>>> a[3][1]
'e'
>>> a[3][-1]
'o'
>>> a[1]
(3+2j)
>>> a[4] = 'hai'
>>> a
[10, 2.8, [1,2,3], 'hello', 'hai']
>>> a[2][1] = 20
>>> a
[10, 2.8, [1,20,3], 'hello', 'hai']
>>> a[3][1] = 'O'
Type error.
>>> a[3] = 67
>>> a
[10, 2.8, [1,20,3], 67, 'hai']
```

> a = [12, [1,2,3, [1,2]], 'hello', [5,6,7], True]

var space	val space
a	0x11
[0x11]	-5 -4 -3 -2 -1 | 12 | 0x13 | 0x33 | 0x44 | 0x55 | 0 1 2 3 4

0x33: -5 -4 -3 -2 -1 | 1 | 2 | 3 | 0x33 | 0 1 2 3

0x13: -2 -1 | 1 | 2 | 0 1

0x44: -3 -2 -1 | 5 | 6 | 7 | 0 1 2

0x55: -3 -2 -1 | T | F | T | 0 1 2

3) Tuple (tuple): is a collection of homogeneous data items of heterogeneous data items enclosed b/w the pair of parenthesis (())

> the syntax to create Tuple is | var = (val1, val2, ... valn) or var = val1, val2, ... valn |

> In Tuple also, the values are separated by comma (,) operator.

> In Tuple, we cannot store a single value directly.

> to store a single value, we have to use syntax. | var = (val,) |

> () is the default value for Tuple collection.

Ex:

```
>>> t = (10,20,30,40)
>>> type(t)
<class 'tuple'>
>>> t
(10,20,30,40)
>>> a = 1,2,3,4,5,6
>>> a
(1,2,3,4,5,6)
>>> type(a)
<class 'tuple'>
```

```
>>> c = ()
>>> bool(a)
True
>>> bool(t)
True
>>> bool(c)
False
>>> c = (67)
>>> type(c)
<class 'int'>
>>> d = ('hello')
>>> type(d)
<class 'str'>
>>> t = (56,)
>>> type(t)
<class 'tuple'>
>>> t = (56)
>>> type(t)
<class 'int'>
>>> s = 9,
>>> s
(9,)
>>> type(s)
<class 'tuple'>
>>> len(s)
1
```

Ex 2: t = (10, 3+8j, 7.9, True, (1,2), [7,8,9,10], 'python')

val space	val space
t	0x11
[0x11]	-7 -6 -5 -4 -3 -2 -1 | 10 | 3+8j | 7.9 | True | 0x22 | 0x33 | 0x44 | 0 1 2 3 4 5 6

0x22: -2 -1 | 1 | 2 | 0 1

0x33: -4 -3 -2 -1 | 7 | 8 | 9 | 10 | 0 1 2 3

0x44: -7 -6 -5 -4 -3 -2 -1 | p | h | y | t | h | o | n | 0 1 2 3 4 5 6

Ex:

```
>>> t[3]
True
>>> t[6]
'python'
>>> t[-7]
'python'
>>> t[0] = 780
TypeError: 'tuple' object does not support item assignment
```

> Since Tuple will not allow the user to modify its original value, & to add a new value & to remove an existing value, we can call it as an 'Immutable Collection'.

> Because of its Immutable nature, Tuple is used in 'Secured data Transmission.'

4) Set (set): Set is an unordered, non-duplicate collection of homogeneous data items & heterogeneous data items enclosed b/w pair of flower braces {}.

> the syntax to create Set is | var = {val1, val2, ... valn} |

> where values should be Immutable (int, float, complex, boolean, str, tuple).

> In case of Set also the values are separated by Comma, operator.

Ex: b = {64, 79.38, 'good morning'}

val space	value space
b	0x11
[0x11]	64, 79, good morning, 38

> In Set, the elements will be arranged in randomly, because of this reason, indexing is not possible.
> Since indexing is not possible, we cannot get an individual values from the Set & it is not possible to modify the original value based on indexing.

Ex:
```
>>> s = {56, 76, 3.14, 'hello', 'hai'}
>>> s
{'hello', 3.14, 'hai', 76, 56}
>>> s = {1, 2, 3, 4, 5}
>>> type(s)
<class 'set'>
>>> s = {'hai', 8, 5, 'hai', 9, 8, 5}
>>> s
{'hai', 9, 8, 5}
>>> a = {12, 8.9, 'hai', 6+7j, True, (1,2,3)}
>>> a
{True, 8.9, 'hai', 12, (1,2,3), (6+7j)}
>>> s = {89, [1,2,3]}
TypeError : unhashable type : 'list'
```

Ex:
```
>>> s = {1, 2, 3}
>>> s.pop()
1
>>> s.pop()
2
>>> s.pop()
3
>>> s
set()
>>> bool(s)
False
>>> bool(1)
True
```

> Set(), is the default value for Set Collection.
> We cannot modify Set collection by using index, but we can modify it by using inbuilt functions.
> add() :- It is a function which is used to add a new value to the existing Set collection & Syntax is

var.add(value)

→ Where value should be immutable.
→ If the value is already present then it will not affect original Set Collection.

> pop() :- It is a function, which is used to remove the 1st value from the given Set. Syntax is

var.pop()

> remove() :- It is a function, which is used to remove specified value from an existing Set collection.
→ If we try to remove the value which is not present, then control will throughout an error. & Syntax is

var.remove(value)

Ex:
```
>>> s = set()
>>> bool(s)
False
>>> a = {11.5, 3, 1.6}
>>> bool(a)
True
>>> a
{1.6, 3, 11.5}
>>> a.add('hello')
>>> a
{'hello', 1.6, 3, 11.5}
>>> a.add(3)
>>> a
{'hello', 1.6, 3, 11.5}
>>> a.add((5,6))
TypeError : unhashable type : 'set'
>>> a.pop()
'hello'
>>> a.pop()
1.6
>>> a.remove(11.5)
>>> a
{3}
>>> a.remove(4)
KeyError : 4
```

> Since Set will remove the duplicate values present in it, we can use it in 'data filtering process'.

---

5. Dictionary (dict) :- Dictionary is a collection of key value pairs enclosed b/w pair of flower braces {}.
> Syntax to create dictionary is `var = {k1:v1, k2:v2 .... kn:vn}`
Where keys should be 'Immutable' & values can be anything.
> In dictionary if we store duplicate keys, then the previous value will get overwritten by the next value of respective key.
> {}, is the default value for dictionary

Ex:
```
>>> d = {'a':1, 'b':2}
>>> type(d)
<class 'dict'>
>>> d = {[1,2]:89}
TypeError : unhashable type : 'list'
>>> a = {'b':8, 'c':9, 'b':'hai'}
>>> a
{'b':'hai', 'c':9}
>>> len(a)
2
>>> d = {}
>>> bool(d)
False
>>> bool(a)
True
```

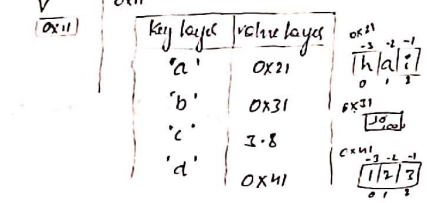
Memory Allocation for dictionary datatype:

> In case of dictionary, the compiler will create 2 layers inside value space. These are key layer & value layer.

> Always the address will be given to key layer & that will get stored with respect to variable name in variable space.

> All the keys & values will get stored into key & value layer respectively.

Ex: v = {'a': 'hai', 'b': 10, 'c': 3.8, 'd': [1,2,3]}



> To access a specific value from dictionary, we can use a syntax
var[key]

> To modify the value & to add a new key value pair, we should use a syntax var[key] = new_value

> To remove the specified key value pair from dictionary, we should use a function called pop() & syntax is var.pop(key)

> Since it is possible to modify the values present inside the dictionary we can call it as 'Mutable collection'

note: In case of dictionary for the compiler the only visible layer is 'key layer' because address will given to key layer.

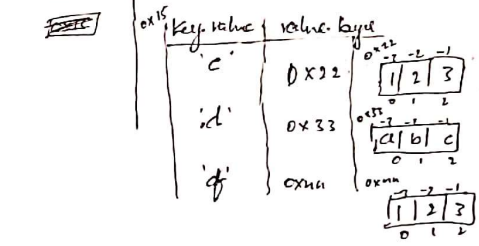
Ex: >>> v = {'a': 'hai', 'b': 10, 'c': 3.8, 'd': [1,2,3]}

```
>>> v['a']
'hai'
>>> v['d']
[1,2,3]
>>> v['a'][1]
'a'
>>> v('hai')
KeyError : 'hai'
```

```
>>> v
{'a': 'hai', 'b': 1000, 'c': 3.8, 'd': [1,2,3]}
>>> v['b'] = 1000
>>> v
{'a': 'hai', 'b': 1000, 'c': 3.8, 'd': [1,2,3]}
>>> v.pop('a')
'hai'
>>> v['e'] = 23
>>> v
{'b': 1000, 'c': 3.8, 'd': [1,2,3], 'e': 23}
>>> v.pop('d')
>>> v
>>> {'b': 1000, 'c': 3.8, 'e': 23}
>>> v['a'] = {67, 81}
>>> v
{'b': 1000, 'c': 3.8, 'e': 23, 'a': {99, 67}}
```

> Indexing is not possible in case of Dictionary.

Ex: a = ['a', 'b', {'c': [1,2,3], 'd': ('a', 'b', 'c'), 'b': [1,2,7]}]



Ex: a = {'a': 1, (1,2,3): [1, 2, 3, 'hello', (1,2,3)], 'hai': ('hi'), 'd': {'a': 1, 'b': 2, 'c': [1,2,3,4,5]}}

=>