






28/9/22

info: way to access/store
Data Structure

→ in general (conventional)

↓
Python

- list
- tuple
- set
- dictionary

- stack 
- queue 
- linked list 
- Tree 
- Graph 

⇒ How many & what are data structures in python

					Access
(user) →	list	[]	Heterogeneous	Dynamic	Indexed
(python) →	tuple	()	"	X (set can't be changed) - static -	"
(uniqueness) →	set	{ }	"	"	custom
(mapping) →	dict	{key : value}	"	"	key ordered
		↑ use symbol to create			

String is not a data structure, it is privileged primitive data type.
 list is most used.

data = {3}

↳ makes dictionary

data = {0}

↳ makes set
[req data in it
to make set]

classmate

Date

Page

→ to create blank set
↳ data = set()

i) List

data = [11, 22, 33, 44]

0 1 2 3 → +ve index
-4 -3 -2 -1 ← -ve index (reverse)

- list is indexed loop
- auto iterated loop.

len() : len(d) → 4

sum() : only number data i.e int / float

min(), max() : only on singular datatype
└ only number
└ only string

sorted() : returns sorted → not permanent

→ auto iteration

```
for item in data:  
    print(item)
```

→ Indexed.

```
for index in range(len(data)):  
    print("at", index, "we have", data[index])
```

→ Reverse a list in one line

data[::-1] - this is temporary.

*inplace → means permanent

→ sorted(d) → ascending by default
→ sorted(d, reverse = True) → descending

→ .sort()

↳ inplace inbuilt method
i.e it changes permanently

d.sort()

d → gives sorted list

for permanently changing using sorted()

d = sorted(d)

d → gives overwritten sorted list only

→ sorts the string with uppercase first & then lowercase

i.e d = ['SHRUTI', 'ANYA', 'siya', 'SIYA']

follows
↘

sorted(d)

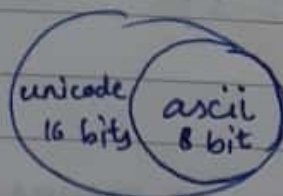
↳ ['ANYA', 'SHRUTI', 'SIYA', 'siya']

* ASCII → ord(i) → gives ascii val of i

* relation bet unicode & ascii

- ascii is a subset of unicode

- unicode is a superset of ascii including it allows 24+ scripting languages.



* Slicing

slice [start : End : step]
 ↓ ↓ ↓
 default default default
 0 len() +1

d [::] → gives full list (default)

d [:: 2] → step 2

d [1 :: 2] → skips 1st ele & steps 2 from 2nd ele

d [2 : 5] → ~~2~~ ele at 2 to ele 4 (stops at 5)

d [: 4] → stops at 4 (prints till 3rd)

d [:: -2] → start from last ele with step 2

* Concatenate or append & copying

d = [1, 2, 3]

d + [10, 20, 30] → d = [1, 2, 3, 10, 20, 30]

d * 2 → makes copies in list

↳ [1, 2, 3, 1, 2, 3]

for i in range(len(d)):

 d[i] = d[i] * 2

print(d)

→ [2, 4, 6, ...]

list.append(x) → add item to the end

list.extend(l) → extend list by appending all items in given list

list.insert(i, x) → insert item at ith position

→ in insert(i, x) if we specify out of range index, then extreme left or extreme right gets inserted.
→ insertion at -1 will be second last position

d.insert(-1, 400)

→ adds 400 to -1 position
- starts -1 from 2nd last position

classmate

Date

Page

- append (x list) inserts a sublist
- extend merges two lists.

list.remove(x) - removes 1st item

list.pop([i]) - removes ith ele or last by default

list.clear() - clear list i.e removes all ele
but do not delete list.

del list → will delete list.

→ index of ith occurring ele

list.index(x)

list.count(x) - count no. of times of x if not present, gives 0

inplace { list.sort()
list.reverse()

[::-1] → temporary

.reverse → permanent

- d.index(11, 1) → key to search, i index onwards.

- d = [11, 22, 33, 44]

→ 420 in d - False

22 in d - True

22 not in d - False

} in & not in used
to check if elements
present or not.

* Nested list

$m = \begin{bmatrix} [10, 20], [30, 40, 50], [60] \end{bmatrix}$

0
1
2

to get 2nd element of 2nd list
 $m[1][1] \rightarrow 40$

2) Tuple ()

— temporary

↓
Static **

↳ Once created can't be changed
 but can't be overwritten.

$t = (10, 20, 30, 40, 50)$

↳ only 2 methods because tuple is static or immutable,
 it does not allow adding
 & removal of data.

- count
- index

Any multi value element if given, then data
 will be converted to tuple and saved.

i.e. $t = 20, 30, 40$

t

↳ o/p $(20, 30, 40) \rightarrow$ tuple

$t = (10, 20, 30)$

$t = (11, 22, 33)$

↳ will overwrite
 & now it will have
 $(11, 22, 33)$

* $t = (10, 20, 30, [11, 22])$

↑

list inside tuple is still dynamic

- addition list can't be added

- but, the elements in this list can be added.

↳ i.e. $t[3].append(100) \rightarrow (10, 20, 30, [11, 22, 100])$

29/9/22

3) Set ~~()~~ ()

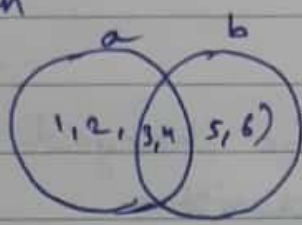
- Dynamic
- Hetrogeneous
- Custom order

$a = \{ \}$

↳ will create dict

empty set - $set()$

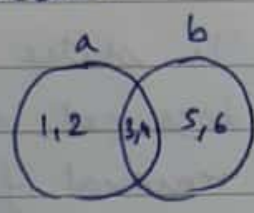
(1) Union



$$a = \{1, 2, 3\} \quad b = \{4, 5, 6\}$$

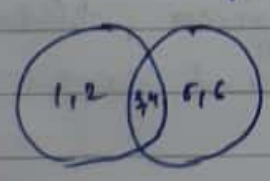
$$a \cup b = \{1, 2, 3, 4, 5, 6\}$$

(2) Intersection



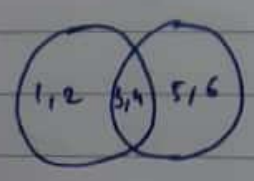
$$a \cap b = \{3, 4\}$$

(3) Symmetric diff.



$$a \Delta b = \{1, 2, 5, 6\}$$

(4) Difference



$$a - b = \{1, 2\}$$

$$b - a = \{5, 6\}$$

* set removes all duplicates from list
→ finds uniqueness

$l = [1, 2, 3, 2, 2, 4, 5, 3, 1]$

$s = \text{set}(l) \rightarrow \{ '1', '2', '3', '4', '5' \}$

→ It does not necessarily prints in order

→ collab prints it in order

i.e. $\text{print}(\text{set}(l)) \rightarrow$ need not be in order.

custom
order

limitation

→ set does not support index based access.

i.e. $sl[0] \rightarrow$ error

↳ index-based access can be done

but,

set can be auto-iterable

for i in sl :

$\text{print}(i) \rightarrow$ need not be in order

- $\text{sum}()$

- $\text{min}()$

- $\text{max}()$

- $\text{.add}(\text{data})$ - add 1 item

- $\text{.update}(\text{set})$ - add set to another set

- $\text{.remove}(\text{data})$ - removes data

- $\text{.discard}(\text{data})$ - removes data without error

* when add existing data again, it does not give error and also does not add duplicates.

- * remove will remove the data & discard also removes the data, but when we remove already removed data, it gives error but discard won't give error.

4) Dictionary { }

- dynamic
- heterogeneous
- duplicate
 - keys not allowed → if duplicate key is used, its value will be replaced.
 - data allowed
- order : key based
- { key: value, key: value }

d = {1: "one", 2: "two", 3: "three"}

d[2] → 'two'

d[4] = "four" → adds 4: "four" after last

d[0.5] = "new" → still adds 0.5: "new" at last
↳ heterogeneous

- * We use dictionary to implement switch case.

> n = int(input("Day no. "))

print(d[n])

→ i/p → 2

o/p → two

* python is interpreter, it will save last updated value.

2

python 3.7 + , dictionary are ordered.

python 3.6 - , dictionary are unordered.

dict

- Get keys → keys() method will return list of keys
d.keys()
 - Get values → values() will return list of values.
d.values()
 - Get items → items() will return each item in a dictionary, as tuples in a list.
d.items()
- dict items([(1, 'one'), (2, 'two'), (3, 'three')])
- ↑ ↑
list tuple
- check if key exists → in k → key.
for k in d: if k in d:
key exists exists
 - update dictionary → var["key"] = value
var.update({key: value})
 - Removing items →
var.pop(key) → deletes → giving key is necessary
del var[key]
var.clear()

- Loops

- print all keys

```
for i in dict :  
    print(i)
```

1
2
3

- print all values

```
for i in dict :  
    print(dict[i])
```

one
two
three

- print both keys & values

```
for k,v in dict.items():  
    print(k,v)
```

1 one
2 two
3 three

if used, like,

```
for i in dict.items():  
    print(i)
```

(1, 'one')
(2, 'two')