Set

A Set is collection of unique items in python. Sets do not allow duplicate items and do not maintain any particular order so it can't be indexed

Unordered - Elements have no defined order. You cannot access elements by index.

Unique Elements - No duplicates allowed. Each element must be distinct.

Mutable - You can add or remove elements after creation.

Immutable Elements - Individual elements inside a set cannot be modified/ replaced

- Note:- set is a immutable but inside values are not mutable

- intialization
- inbuilt functions
  - min
  - max
  - type
  - print
  - len
  - sum
  - sorted
  - reversed
- index
- mutable vs immutable
- concatenation
- slicing

##

- Methods of Set

In [170…
```
s1 = {10,20,30}
s1
```

Out[170…
```
{10, 20, 30}
```

In [2]:
```
s2 = {"hello", "Hi"}
s2
```

Out[2]:
```
{'Hi', 'hello'}
```

In [3]:
```
s3 = {10,20,30,"A",'B'}
s3
```

Out[3]:
```
{10, 20, 30, 'A', 'B'}
```

In [7]:
```
s4 = {10,12.2,'Apple', True, False,20+9j}
s4

# any type of data valid
```

Out[7]:
```
{(20+9j), 10, 12.2, 'Apple', False, True}
```

In [8]:
```
s5 = {10,20,10,20,32}
s5

# Duplicates are not allowed
```

Out[8]:
```
{10, 20, 32}
```

In [9]:
```
s6 = {10,20,{100,200}}
s6
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 s6 = {10,20,{100,200}}
      2 s6

TypeError: unhashable type: 'set'
```

In [10]:
```
s7 = {10,20,30,[100]}
s7
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 s7 = {10,20,30,[100]}
      2 s7

TypeError: unhashable type: 'list'
```

In [11]:
```
s8 = {10,20,'Hello',(10,20,30)}
s8
```

Out[11]:
```
{(10, 20, 30), 10, 20, 'Hello'}
```

- A set can contain immutable types like integers, strings, tuples, and frozensets.
- A set cannot contain mutable types like lists or dictionaries, or another regular set.
- For nested sets, you must use frozenset.

- Duplicates are not allowed in set.
- Any type of data valid int,float,string,complex,boolean

In [166…
```
s9 = set()
print(s9)
type(s9)
# empty set allowed
```

```
set()
```

Out[166… set

In [168…
```
s10 = {10,20,(23,56,78)}
max(s10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[168], line 2
      1 s10 = {10,20,(23,56,78)}
----> 2 max(s10)

TypeError: '>' not supported between instances of 'tuple' and 'int'
```

In [29]:
```
t=(_)
print(t)
```

```
<class 'type'>
```

Inbuilt

max and min

In [34]:
```
max(s1),max(s2),max(s5)
```

Out[34]: (30, 'hello', 32)

In [36]:
```
min(s1),min(s2),min(s5)
```

Out[36]: (10, 'Hi', 10)

- Not valid in s3,s4,s6,s8,s9

type

In [37]:
```
type(s1),type(s2)
```

Out[37]: (set, set)

len

In [74]:
```
len(s1), len(s2), len(s3), len(s4), len(s5), len(s8), len(s9)
```

Out[74]: (3, 2, 5, 6, 3, 4, 0)

sum

In [46]:
```
sum(s1)
```

Out[46]: 60

In [47]:
```
sum(s1, start=100)
```

Out[47]: 160

In [53]:
```
sum(s3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[53], line 1
----> 1 sum(s3)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

sorted

In [171…
```
sorted(s1)
```

Out[171… [10, 20, 30]

In [51]:
```
sorted(s3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[51], line 1
----> 1 sorted(s3)

TypeError: '<' not supported between instances of 'int' and 'str'
```

In [57]:
```
l1={'Nest','Mango','Zebra','Elephant','Apple'}
sorted(l1)
```

Out[57]: ['Apple', 'Elephant', 'Mango', 'Nest', 'Zebra']

In [58]:
```
l1={'Nest','Mango','Zebra','Elephant','Apple'}
sorted(l1, key=len)
```

Out[58]: ['Nest', 'Zebra', 'Mango', 'Apple', 'Elephant']

In [59]:
```
l1={'Nest','Mango','Zebra','Elephant','Apple'}
sorted(l1,reverse=True)
```

Out[59]: ['Zebra', 'Nest', 'Mango', 'Elephant', 'Apple']

In [60]:
```
l1={'Nest','Mango','Zebra','Elephant','Apple'}
sorted(l1,key=len, reverse=True)
```

Out[60]: ['Elephant', 'Zebra', 'Mango', 'Apple', 'Nest']

reversed

In [56]:
```
reversed(s2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[56], line 1
----> 1 reversed(s2)

TypeError: 'set' object is not reversible
```

- **Set are not reversible**

index

In [63]:
```
s4 = {(20+9j), 10, 12.2, 'Apple', False, True}
s4[0]
```

Out[63]:
```
{(20+9j), 10, 12.2, 'Apple', False, True}
```

In [64]:
```
4[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[64], line 1
----> 1 s4[0]

TypeError: 'set' object is not subscriptable
```

- In Python, sets are unordered collections. This means that the elements of a set do not have a specific index, unlike lists or tuples.
- thats why we are not use index and slicing

slicing

In [65]:
```
s4[0:]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[65], line 1
----> 1 s4[0:]

TypeError: 'set' object is not subscriptable
```

mutable vs immuatvle

In [66]:
```
s1[0] = 10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[66], line 1
----> 1 s1[0] = 10

TypeError: 'set' object does not support item assignment
```

- immutable

concatenation

In [67]:
```
s1 + s2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[67], line 1
----> 1 s1 + s2

TypeError: unsupported operand type(s) for +: 'set' and 'set'
```

- Sets are unordered collections of unique elements. The + operator is not defined for sets because sets do not maintain order, and concatenation (like with lists) doesn't apply to them.

not allowed

- In set Indexing, Slicing, concatenation, not allowed
- because sets elements are not ordered
- Set is a Immutable data type
- In Set reversing not allowed

In [70]:
```
dir(s1)
```

Out[70]:
```
['__and__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__iand__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__isub__',
 '__iter__',
 '__ixor__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__rand__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__ror__',
 '__rsub__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__xor__',
 'add',
 'clear',
 'copy',
 'difference',
 'difference_update',
 'discard',
 'intersection',
 'intersection_update',
 'isdisjoint',
 'issubset',
 'issuperset',
 'pop',
 'remove',
 'symmetric_difference',
 'symmetric_difference_update',
 'union',
 'update']
```

Methods In Set

- add
- clear
- copy
- difference
- difference_update
- discard
- intersection
- intersection_update
- isdisjoint
- issubset
- issuperset
- pop
- remove
- symmetric_difference
- symmetric_difference_update
- union
- update

add

In [76]:
```
s2.add("hello hi")
s2
```

Out[76]:
```
{'Hi', 'hello', 'hello hi'}
```

In [173…
```
s2.add(75)
```

In [80]:
```
s2.add("days")
s2
```

Out[80]:
```
{75, 'Hi', 'days', 'hello', 'hello hi'}
```

In [85]:
```
s2.add()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[85], line 1
----> 1 s2.add()

TypeError: set.add() takes exactly one argument (0 given)
```

update

In [128…
```
s1 = {75, 'Hi', 'days', 'hello', 'hello hi'}
s1
```

Out[128…
```
{75, 'Hi', 'days', 'hello', 'hello hi'}
```

In [179…
```
s1.update('hi')
s1
```

Out[179… {10, 20, 30, 'are', 'h', 'how', 'i', 'you'}

In [131…
```
s1.update(10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[131], line 1
----> 1 s1.update(10)

TypeError: 'int' object is not iterable
```

- 10 is an integer, and integers are not iterable. The update() method expects an iterable (like a list, set, or tuple), and since an integer is not iterable, it raises an error.

In [176…
```
s1.update([10,20,30]) # list giving
s1 # added list elements only not whole list as it is
```

Out[176… {10, 20, 30}

In [177…
```
s1.update(('how', 'are', 'you')) # tuple
s1
```

Out[177… {10, 20, 30, 'are', 'how', 'you'}

In [180…
```
s1.update({10: 'school'}) # dict
s1
```

Out[180… {10, 20, 30, 'are', 'h', 'how', 'i', 'you'}

In [181…
```
s1.update({1012332}) #set
s1
```

Out[181… {10, 1012332, 20, 30, 'are', 'h', 'how', 'i', 'you'}

clear

In [117…
```
s1 = {75, 'Hi', 'days', 'hello', 'hello hi'}
s1
```

Out[117… {75, 'Hi', 'days', 'hello', 'hello hi'}

In [119…
```
s1.clear()
s1 # remaning only empty set & no error
```

Out[119… set()

Copy

In [120…
```
s1 = {75, 'Hi', 'days', 'hello', 'hello hi'}
s1
```

Out[120… {75, 'Hi', 'days', 'hello', 'hello hi'}

In [185…
```
s2 = set()
s2 = s1.copy()
```

```
s2
```

Out[185… {10, 1012332, 20, 30, 'are', 'h', 'how', 'i', 'you'}

In [182…
```
s3 = set()
s3 = s1.copy()
s3
```

Out[182… {10, 1012332, 20, 30, 'are', 'h', 'how', 'i', 'you'}

remove

In [186…
```
s2.remove(10)
s2
```

Out[186… {1012332, 20, 30, 'are', 'h', 'how', 'i', 'you'}

In [84]:
```
s2.remove()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[84], line 1
----> 1 s2.remove()

TypeError: set.remove() takes exactly one argument (0 given)
```

In [188…
```
s2.remove('a')
S2
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[188], line 1
----> 1 s2.remove('a')
      2 S2

KeyError: 'a'
```

- Remove an element from a set; it must be a member.

discard

In [86]:
```
s2
```

Out[86]: {'Hi', 'days', 'hello', 'hello hi'}

In [87]:
```
s2.discard()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[87], line 1
----> 1 s2.discard()

TypeError: set.discard() takes exactly one argument (0 given)
```

In [187…
```
s2.discard('not')
s2
```

Out[187… {1012332, 20, 30, 'are', 'h', 'how', 'i', 'you'}

In [90]:
```
s2.discard('Hi')
s2
```

Out[90]: {'days', 'hello', 'hello hi'}

pop

In [162…
```
s1 = {30, 'e', 'h', 'how', 'i', 'r', 'u', 'you'}
s1
```

Out[162… {30, 'e', 'h', 'how', 'i', 'r', 'u', 'you'}

In [163…
```
s1.pop()
```

Out[163… 'how'

In [164…
```
s1.pop('h')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[164], line 1
----> 1 s1.pop('h')

TypeError: set.pop() takes no arguments (1 given)
```

pop

- It will remove randomly elements inside the sets.
- If set is empty it will throw the error
- No need to pass any value or index inside pop it will throw error

remove vs discard

- if we pass wrong value inside remove it will **show error**
- but if we pass wrong value inside discard **no output and no error**

union

- Combines elements from two sets, removing duplicates.

In [94]:
```
s1 = {10,20,30,20,30,40,50,60}
s2 = {10,'hello', 'python', 10, 100}

s1.union(s2)
s1
```

Out[94]: {10, 20, 30, 40, 50, 60}

In [96]:
```
s1 = {'hai', 'how', 'are', 'you'}
s2 = {10,'how', 'are', 10, 100}
```

```
s1.union(s2)
```

Out[96]: {10, 100, 'are', 'hai', 'how', 'you'}

In [101…
```
s1 = s1 | s2 # It works like same union
s1
```

Out[101… {10, 100, 'are', 'hai', 'how', 'you'}

intersection

- Includes only elements presents in both sets

In [106…
```
s1 = {'hai', 'how', 'are', 'you'}
s2 = {10,'how', 'are', 10, 100}

s1.intersection(s2)
```

Out[106… {'are', 'how'}

In [108…
```
s1 & s2 # It works like a intersection
```

Out[108… {'are', 'how'}

difference

- Elements present in the first set but not in the second

In [98]:
```
s1.difference(s2)
```

Out[98]: {'hai', 'you'}

In [105…
```
s1 - s2 # It works like a difference
```

Out[105… {'hai', 'you'}

symmetric difference

- Elements in either set, but not in both.

In [109…
```
s1 = {1,2,3,4,5,}
s2 = {3,4,5,6,7,1}

s1.symmetric_difference(s2)
```

Out[109… {2, 6, 7}

In [111…
```
s1 = {1,2,3,4,5,}
s2 = {3,4,5,6,7,1}

s1 ^ s2
```

Out[111… {2, 6, 7}

Set Iteration

- We are not able to use while looop in set
- because during while loop required a index number
- in set has no index

In [112…
```
num = {12,23,43,54,6,76,8,9,67,4,34,3423,34}
for i in num:
    print(i)
```

```
34
67
4
6
8
9
43
12
76
54
23
3423
```

Set Comprehension

In [113…
```
s1 = {i for i in num}
s1
```

Out[113… {4, 6, 8, 9, 12, 23, 34, 43, 54, 67, 76, 3423}

In [114…
```
squ = {i**2 for i in range(1,10)}
squ
```

Out[114… {1, 4, 9, 16, 25, 36, 49, 64, 81}

In [116…
```
cub = {i**3 for i in range(1,10)}
cub
```

Out[116… {1, 8, 27, 64, 125, 216, 343, 512, 729}

- **Best use case of Set is he not allowed duplicates**
- **Set Operations**
  - union, intersection, difference
- **Data Analysis**
  - Useful in scenarios requiring unique items, such as tags, categories or unique identifires

In [ ]: