superset & DICTIONARY

subset ### dict.keys() dict.copy()

disjoint ### dict.fromkeys(iterable, value)

dict.clear() dict.get(key, default)

dict.items()

dict.values()

dict.pop(key,default)

dict.popitem()

dict.setdefault

dict.update

In [15]:
```
s11={1,2,3,4,5,6,7,8,9}
s12={3,4,5,6,7,8}
s13={10,20,30,40}
```

In [17]:
```
s12.issubset(s11)
```

Out[17]: True

In [19]:
```
s11.issuperset(s12)
```

Out[19]: True

In [21]:
```
s13.isdisjoint(s11,s12)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 1
----> 1 s13.isdisjoint(s11,s12)

TypeError: set.isdisjoint() takes exactly one argument (2 given)
```

In [23]:
```
s13.isdisjoint(s11)
```

Out[23]: True

In [25]:
```
s13.isdisjoint(s12)
```

Out[25]: True

In [27]:
```
s14={1,2,3,4}
s15={5,6,7}
s16={10,20,30}
```

In [29]:
```
s13.issubset(s14)
```

Out[29]: False

In [31]:
```
s12.issubset(s13)
```

Out[31]: False

In [33]:
```
s16.isdisjoint(s14)
```

Out[33]: True

In [35]:
```
s16.isdisjoint(s15)
```

Out[35]: True

In [37]:
```
s14={1,2,3,4}
s15={5,6,7}
s16={5,20,30}
```

In [39]:
```
s16.isdisjoint(s15)
```

Out[39]: False

In [45]:
```
s16.isdisjoint(s14)
```

Out[45]: True

In [ ]:
```
s14={1,2,3,4}
s15={5,6,7}
s16={10,20,30}
```

In [51]:
```
s14.add(5)
```

In [53]:
```
s14
```

Out[53]: {1, 2, 3, 4, 5}

In [57]:
```
s15.clear()
```

In [59]:
```
s15.union
```

Out[59]: set()

In [63]:
```
s17=s16.copy()
s17
```

Out[63]: {5, 20, 30}

In [69]:
```
print(s14)
print(s16)
```

```
{1, 2, 3, 4, 5}
{20, 5, 30}
```

In [71]:
```
s16.difference(s14)
```

Out[71]: {20, 30}

In [75]:
```
s14.difference(s16)
```

Out[75]: {1, 2, 3, 4}

In [81]:
```
s15
```

Out[81]: set()

In [95]:
```
s14
```

Out[95]: {1, 2, 3, 4, 5}

In [91]:
```
s14.symmetric_difference(s15)
```

Out[91]: {1, 2, 3, 4, 5}

In [93]:
```
s15
```

Out[93]: set()

In [97]:
```
s14.discard(4)
```

In [99]:
```
s14
```

Out[99]: {1, 2, 3, 5}

In [101…
```
s14.discard(6)#if element not given,even though it will not give error
s14
```

Out[101… {1, 2, 3, 5}

In [103…
```
s16.intersection(s14)
```

Out[103… {5}

In [119…
```
s16.pop()
s16
```

Out[119… set()

In [117…
```
s17=(1,2,3,4)#index and slicing not allowed
s17.pop(0)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[117], line 2
      1 s17=(1,2,3,4)
----> 2 s17.pop(0)

AttributeError: 'tuple' object has no attribute 'pop'
```

In [121…
```
s16
```

Out[121… set()

In [123…
```
s14
```

Out[123… {1, 2, 3, 5}

In [125…
```
s15
```

Out[125… set()

In [127…
```
s16==s15
```

Out[127… True

In [129…
```
s16.update(s14)
s16
```

Out[129… {1, 2, 3, 5}

In [131…
```
s15.update(1)
s15
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[131], line 1
----> 1 s15.update(1)
      2 s15

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

In [133…
```
s16.remove(1)
s16
```

Out[133…
```
{2, 3, 5}
```

In [137…
```
s15
```

Out[137…
```
set()
```

In [135…
```
s15.add(s14)#we cant add one set to another because set is unhashable.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[135], line 1
----> 1 s15.add(s14)

TypeError: unhashable type: 'set'
```

In [139…
```
s16.
```

```
  Cell In[139], line 1
    s16.
        ^
SyntaxError: invalid syntax
```

In [141…
```
s17
```

Out[141…
```
(1, 2, 3, 4)
```

In [143…
```
s14
```

Out[143…
```
{1, 2, 3, 5}
```

In [165…
```
s18={1,2,3,4,5,6}
s14.difference(s18)
```

Out[165…
```
set()
```

In [167…
```
s18
```

Out[167…
```
{1, 2, 3, 4, 5, 6}
```

In [169…
```
s18.symmetric_difference(s14)
```

Out[169…
```
{4, 6}
```

In [171…
```
s18
```

Out[171…
```
{1, 2, 3, 4, 5, 6}
```

In [173…
```
s19={1,2,3,}
s18.difference(s19)
```

Out[173…
```
{4, 5, 6}
```

In [175…
```
s18
```

Out[175…
```
{1, 2, 3, 4, 5, 6}
```

In [177…
```
s19
```

Out[177…
```
{1, 2, 3}
```

In [179…
```
s18.symmetric_difference(s19)
```

Out[179…
```
{4, 5, 6}
```

In [183…
```
s19={1,2,3,7,8}
s18.symmetric_difference(s19)
```

Out[183…
```
{4, 5, 6, 7, 8}
```

In [185…
```
print(s18)
print(s19)
```

```
{1, 2, 3, 4, 5, 6}
{1, 2, 3, 7, 8}
```

In [193…
```
s18
```

Out[193…
```
{1, 2, 3, 4, 5, 6}
```

In [195…
```
s18.symmetric_difference_update(s19)#it update the original set.
```

In [198…
```
s18
```

Out[198…
```
{4, 5, 6, 7, 8}
```

In [200…
```
s18
```

Out[200…
```
{4, 5, 6, 7, 8}
```

In [202…
```
print(s15)
print(s16)
print(s17)
print(s18)
```

```
set()
{2, 3, 5}
(1, 2, 3, 4)
{4, 5, 6, 7, 8}
```

In [204…
```
s16.update(s17)
```

In [206…
```
s16
```

Out[206…
```
{1, 2, 3, 4, 5}
```

In [208…
```
s16
```

Out[208…
```
{1, 2, 3, 4, 5}
```

In [210…
```
s17
```

Out[210…
```
(1, 2, 3, 4)
```

In [216…
```
s16.difference_update(s17)#its change the original set
```

In [218…
```
s16
```

Out[218…
```
{5}
```

In [220…
```
s16
```

Out[220…
```
{5}
```

In [222…
```
s18
```

Out[222…
```
{4, 5, 6, 7, 8}
```

In [224…
```
s14
```

Out[224…
```
{1, 2, 3, 5}
```

In [226…
```
s18.intersection(s14)
```

Out[226…
```
{5}
```

In [228…
```
s18
```

Out[228…
```
{4, 5, 6, 7, 8}
```

In [230…
```
s18.intersection_update(s14)#it modify the original set
```

In [232…
```
s18
```

Out[232…
```
{5}
```

In [234…
```
s18
```

Out[234… {5}

In [236…
```
for i in s14:
    print(i)
```

```
1
2
3
5
```

In [242…
```
for i in enumerate (s14):
    print(i)
```

```
(0, 1)
(1, 2)
(2, 3)
(3, 5)
```

In [244…
```
sum(s14)
```

Out[244… 11

In [246…
```
max(s14)
```

Out[246… 5

In [248…
```
min(s14)
```

Out[248… 1

In [250…
```
len(s14)
```

Out[250… 4

In [254…
```
1 in s14
```

Out[254… True

In [265…
```
sorted(s14)
```

Out[265… [1, 2, 3, 5]

In [269…
```
s={1,3,6,4,2,7,9}
s
```

Out[269… {1, 2, 3, 4, 6, 7, 9}

In [275…
```
sorted(s)
```

Out[275… [1, 2, 3, 4, 6, 7, 9]

In [277…
```
s.add(5)
```

In [279…
```
s
```

Out[279…
```
{1, 2, 3, 4, 5, 6, 7, 9}
```

In [285…
```
s1={2,1,5,4,7,9}
sorted(s1)
```

Out[285…
```
[1, 2, 4, 5, 7, 9]
```

In [287…
```
reversed(s1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[287], line 1
----> 1 reversed(s1)

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

In [289…
```
range(s1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[289], line 1
----> 1 range(s1)

TypeError: 'set' object cannot be interpreted as an integer
```

In [293…
```
abs(s1)
```

```
  Cell In[293], line 1
    abs(s1):
           ^
SyntaxError: invalid syntax
```

In [295…
```
round(s1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[295], line 1
----> 1 round(s1)

TypeError: type set doesn't define __round__ method
```

In [297…
```
zip(s1)
```

Out[297…
```
<zip at 0x2bfc19d1a40>
```

In [299…
```
id(s1)
```

Out[299…
```
3022617724960
```

In [301…
```
help(s1)
```

```
Help on set object:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
```

```
|      Return self!=value.
|
|  __or__(self, value, /)
|      Return self|value.
|
|  __rand__(self, value, /)
|      Return value&self.
|
|  __reduce__(...)
|      Return state information for pickling.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(...)
|      S.__sizeof__() -> size of S in memory, in bytes
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  add(...)
|      Add an element to a set.
|
|      This has no effect if the element is already present.
|
|  clear(...)
|      Remove all elements from this set.
|
|  copy(...)
|      Return a shallow copy of a set.
|
|  difference(...)
|      Return the difference of two or more sets as a new set.
|
|      (i.e. all elements that are in this set but not the others.)
|
|  difference_update(...)
|      Remove all elements of another set from this set.
|
|  discard(...)
|      Remove an element from a set if it is a member.
|
|      Unlike set.remove(), the discard() method does not raise
|      an exception when an element is missing from the set.
```

```
|
|  intersection(...)
|      Return the intersection of two sets as a new set.
|
|      (i.e. all elements that are in both sets.)
|
|  intersection_update(...)
|      Update a set with the intersection of itself and another.
|
|  isdisjoint(...)
|      Return True if two sets have a null intersection.
|
|  issubset(self, other, /)
|      Test whether every element in the set is in other.
|
|  issuperset(self, other, /)
|      Test whether every element in other is in the set.
|
|  pop(...)
|      Remove and return an arbitrary set element.
|      Raises KeyError if the set is empty.
|
|  remove(...)
|      Remove an element from a set; it must be a member.
|
|      If the element is not a member, raise a KeyError.
|
|  symmetric_difference(...)
|      Return the symmetric difference of two sets as a new set.
|
|      (i.e. all elements that are in exactly one of the sets.)
|
|  symmetric_difference_update(...)
|      Update a set with the symmetric difference of itself and another.
|
|  union(...)
|      Return the union of sets as a new set.
|
|      (i.e. all elements that are in either set.)
|
|  update(...)
|      Update a set with the union of itself and others.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(...)
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs)
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
```

```
|  Data and other attributes defined here:
|
|  __hash__ = None
```

Dictionary

In [316…
```
d={}
d
```

Out[316…
```
{}
```

In [318…
```
type(d)
```

Out[318…
```
dict
```

In [320…
```
d1={1:"0ne",2:"wo",3:"three"}
d1
```

Out[320…
```
{1: '0ne', 2: 'wo', 3: 'three'}
```

In [322…
```
d1.keys()
```

Out[322…
```
dict_keys([1, 2, 3])
```

In [324…
```
d1.values()
```

Out[324…
```
dict_values(['0ne', 'wo', 'three'])
```

In [326…
```
d2=d1.copy()
d2
```

Out[326…
```
{1: '0ne', 2: 'wo', 3: 'three'}
```

In [328…
```
d2.clear()
```

In [330…
```
d2
```

Out[330…
```
{}
```

In [332…
```
d2.items()
```

Out[332…
```
dict_items([])
```

In [336…
```
d1.get(1)
```

Out[336…
```
'0ne'
```

In [402…
```
keys={"sk","pk","a"}
value=[1,2,3]
```

```
d3=dict.fromkeys(keys,value)
d3
```

Out[402…
```
{'a': [1, 2, 3], 'pk': [1, 2, 3], 'sk': [1, 2, 3]}
```

In [404…
```
value.append(50)
d3
```

Out[404…
```
{'a': [1, 2, 3, 50], 'pk': [1, 2, 3, 50], 'sk': [1, 2, 3, 50]}
```

In [406…
```
range(10)
```

Out[406…
```
range(0, 10)
```

In [412…
```
len(range(10))
```

Out[412…
```
10
```

In [408…
```
list(range(0,10))
```

Out[408…
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [414…
```
tuple(range(10))
```

Out[414…
```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

In [348…
```
d1.fromkeys("f",4)
```

Out[348…
```
{'f': 4}
```

In [416…
```
set(range(10))
```

Out[416…
```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [418…
```
dict(range(10))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[418], line 1
----> 1 dict(range(10))

TypeError: cannot convert dictionary update sequence element #0 to a sequence
```

In [364…
```
d1
```

Out[364…
```
{2: 'wo', 3: 'three'}
```

In [368…
```
d1.pop(2)
```

Out[368…
```
'wo'
```

In [376…
```
d2={"ONE":1,"TWO":2,"THREE":3}
d2.pop("ONE")
```

Out[376… 1

In [378…
```
d2
```

Out[378… {'TWO': 2, 'THREE': 3}

In [388…
```
d2.popitem()#it take no argument
```

Out[388… ('TWO', 2)

In [384…
```
d2
```

Out[384… {'TWO': 2}

In [420…
```
list(range(10,20))
```

Out[420… [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

In [424…
```
list(range(10,20,2))
```

Out[424… [10, 12, 14, 16, 18]

In [426…
```
r=range(1,10)
r
```

Out[426… range(1, 10)

In [428…
```
for i in r:
    print(i)
```

```
1
2
3
4
5
6
7
8
9
```

In [442…
```
d3
```

Out[442… {'a': [1, 2, 3, 50], 'pk': [1, 2, 3, 50], 'sk': [1, 2, 3, 50]}

In [436…
```
d4={'one':1,'two':2,'three':3}
d4
```

Out[436… {'one': 1, 'two': 2, 'three': 3}

In [440…
```
d4.update(d3)
d4
```

Out[440…
```
{'one': 1,
 'two': 2,
 'three': 3,
 'a': [1, 2, 3, 50],
 'pk': [1, 2, 3, 50],
 'sk': [1, 2, 3, 50]}
```

In [458…
```
d5={'one':1,'two':2,'three':3,'four':4}
d5
```

Out[458…
```
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

In [466…
```
d5.setdefault('one',100)
```

Out[466…
```
1
```

In [464…
```
d5.get('one')
```

Out[464…
```
1
```

In [468…
```
for i,key in enumerate (d5):
    print(i,key)
```

```
0 one
1 two
2 three
3 four
```

In [472…
```
'one' in d5
```

Out[472…
```
True
```

In [ ]: