- super set
- sub set
- disjoint set

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

In [7]:
```
s12.issubset(s11)
```

Out[7]: True

In [9]:
```
s11.issuperset(s12)
```

Out[9]: True

In [13]:
```
s12.isdisjoint(s13)
```

Out[13]: True

In [15]:
```
sum(s12)
```

Out[15]: 33

In [17]:
```
min(s12)
```

Out[17]: 3

dictionary

In [22]:
```
d = {1:'one', 2:'two', 3:'three'}
```

In [26]:
```
print(type(d))
```

```
<class 'dict'>
```

In [28]:
```
d
```

Out[28]: {1: 'one', 2: 'two', 3: 'three'}

In [30]:
```
d.keys()
```

Out[30]: dict_keys([1, 2, 3])

In [32]:
```
d.values()
```

Out[32]: dict_values(['one', 'two', 'three'])

In [39]:
```
d.items()
```

Out[39]: dict_items([(1, 'one'), (2, 'two'), (3, 'three')])

In [50]:
```
d1=d.copy()
```

In [52]:
```
d1
```

Out[52]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [54]:
```
d1[3]
```

Out[54]:
```
'three'
```

In [58]:
```
k = {1,2,3,4}
v = {1,4,9,16}
d2 = dict.fromkeys(k,v)
d2
```

Out[58]:
```
{1: {1, 4, 9, 16}, 2: {1, 4, 9, 16}, 3: {1, 4, 9, 16}, 4: {1, 4, 9, 16}}
```

In [60]:
```
k = {1,2,3,4}
v = (1,4,9,16)
d3 = dict.fromkeys(k,v)
d3
```

Out[60]:
```
{1: (1, 4, 9, 16), 2: (1, 4, 9, 16), 3: (1, 4, 9, 16), 4: (1, 4, 9, 16)}
```

In [74]:
```
k = (1,2,3,4)
v = [1,4,9,16]
d4 = dict.fromkeys(k,v)
d4
```

Out[74]:
```
{1: [1, 4, 9, 16], 2: [1, 4, 9, 16], 3: [1, 4, 9, 16], 4: [1, 4, 9, 16]}
```

In [76]:
```
v.append(10)
d4
```

Out[76]:
```
{1: [1, 4, 9, 16, 10],
 2: [1, 4, 9, 16, 10],
 3: [1, 4, 9, 16, 10],
 4: [1, 4, 9, 16, 10]}
```

In [72]:
```
k = [1,2,3,4]
v = 10
d5 = dict.fromkeys(k,v)
d5
```

Out[72]:
```
{1: 10, 2: 10, 3: 10, 4: 10}
```

In [80]:
```
k = [1,2,3,4]
v = [10,20,30,40]
d5 = {key:value for key,value in zip(k,v)}
d5
```

Out[80]:
```
{1: 10, 2: 20, 3: 30, 4: 40}
```

In [82]:
```
k = {1,2,3,4}
v = (10,20,30,40)
d5 = {key:value for key,value in zip(sorted(k),v)}
d5
```

Out[82]:
```
{1: 10, 2: 20, 3: 30, 4: 40}
```

In [84]:
```
list(range(10))
```

Out[84]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [86]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [94]:
```
list(range(3,9))
```

Out[94]:
```
[3, 4, 5, 6, 7, 8]
```

In [88]:
```
list(range(4,10,2))
```

Out[88]:
```
[4, 6, 8]
```

In [90]:
```
tuple(range(2,20,3))
```

Out[90]:
```
(2, 5, 8, 11, 14, 17)
```

In [96]:
```
var_name = 5
var_name
```

Out[96]:
```
5
```

In [114…
```
a={1,2,3}
b={2,3,4}
c={7,8,9}
```

In [116…
```
a.issuperset(b)
```

Out[116…
```
False
```

In [118…
```
b.issubset(a)
```

Out[118…
```
False
```

In [130…
```
myset3 = {10,20, "Hola", (11, 22, 32)} # Mixed datatypes
myset3
```

Out[130…
```
{(11, 22, 32), 10, 20, 'Hola'}
```

In [150…
```
myset4 = {10,20, "Hola",(1,2,3,4),(1,2,3,4,5)} # inside set only datastructure allowed is tuple
myset4
```

Out[150…
```
{(1, 2, 3, 4), (1, 2, 3, 4, 5), 10, 20, 'Hola'}
```

In [154…
```
my_set1 = set('one' , 'two' , 'three' , 'four')
my_set1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[154], line 1
----> 1 my_set1 = set('one' , 'two' , 'three' , 'four')
      2 my_set1

TypeError: set expected at most 1 argument, got 4
```

In [156…
```
my_set1 = set(('one' , 'two' , 'three' , 'four'))
my_set1
```

Out[156…
```
{'four', 'one', 'three', 'two'}
```

In [164…
```
l=list('1234')
l
```

Out[164…
```
['1', '2', '3', '4']
```

In [166…
```
l=list((1,2,3,4))
l
```

Out[166…
```
[1, 2, 3, 4]
```

In [168…
```
l=list(1234)
l
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[168], line 1
----> 1 l=list(1234)
      2 l

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

In [170…
```
a={1,2,3,4}
b={3,4,5,6}
a&b
```

Out[170…
```
{3, 4}
```

In [172…
```
a.intersection(b)
```

Out[172…
```
{3, 4}
```

In [174…
```
a
```

Out[174…
```
{1, 2, 3, 4}
```

In [176…
```
a.intersection_update(b)
```

In [178…
```
a
```

Out[178…
```
{3, 4}
```

In [182…
```
list(enumerate(b))
```

Out[182…
```
[(0, 3), (1, 4), (2, 5), (3, 6)]
```

In [184…
```
sorted(b)
```

Out[184…
```
[3, 4, 5, 6]
```

In [186…
```
sorted(b,reverse=True)
```

Out[186…
```
[6, 5, 4, 3]
```

In [198…
```
mydict = {1 : 'one', 'a': 'two', 3 : 'three', 4 : [1,2,3,4], 'b' : (1,2,3,4)}
mydict
```

Out[198…
```
{1: 'one', 'a': 'two', 3: 'three', 4: [1, 2, 3, 4], 'b': (1, 2, 3, 4)}
```

In [200…
```
mydict. keys()
```

Out[200…
```
dict_keys([1, 'a', 3, 4, 'b'])
```

In [202…
```
mydict.values()
```

Out[202…
```
dict_values(['one', 'two', 'three', [1, 2, 3, 4], (1, 2, 3, 4)])
```

In [204…
```
mydict.items()
```

Out[204…
```
dict_items([(1, 'one'), ('a', 'two'), (3, 'three'), (4, [1, 2, 3, 4]), ('b', (1, 2, 3, 4))])
```

In [206…
```
k = {1,2,3,4}
dic = dict.fromkeys(k)
dic
```

Out[206…
```
{1: None, 2: None, 3: None, 4: None}
```

In [210…
```
k = {1,2,3,4,5}
v = {10}
d = dict.fromkeys(k,v)
d
```

Out[210…
```
{1: {10}, 2: {10}, 3: {10}, 4: {10}, 5: {10}}
```

In [220…
```
k = [1,2,3,4]
v =['one','two','three','four']
d = {key : value for key,value in zip(k,v)}
d
```

Out[220…
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [226…
```
d[4]
```

Out[226…
```
'four'
```

In [228…
```
d.get(4)
```

Out[228…
```
'four'
```

In [230…
```
mydict
```

Out[230…
```
{1: 'one', 'a': 'two', 3: 'three', 4: [1, 2, 3, 4], 'b': (1, 2, 3, 4)}
```

In [232…
```
d
```

Out[232…
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [234…
```
d[4] = '4'
```

In [236…
```
d
```

Out[236…
```
{1: 'one', 2: 'two', 3: 'three', 4: '4'}
```

In [242…
```
d.update(mydict)
d
```

Out[242…
```
{1: 'one',
 2: 'two',
 3: 'three',
 4: [1, 2, 3, 4],
 'a': 'two',
 'b': (1, 2, 3, 4)}
```

In [244…
```
d.pop('a')
d
```

Out[244…
```
{1: 'one', 2: 'two', 3: 'three', 4: [1, 2, 3, 4], 'b': (1, 2, 3, 4)}
```

In [246…
```
d.popitem()
```

Out[246…
```
('b', (1, 2, 3, 4))
```

In [248…
```
d
```

Out[248…
```
{1: 'one', 2: 'two', 3: 'three', 4: [1, 2, 3, 4]}
```

In [250…
```
del[d[1]]
```

In [252…
```
d
```

Out[252…
```
{2: 'two', 3: 'three', 4: [1, 2, 3, 4]}
```

In [254…
```
d.clear()
```

In [256…
```
d
```

Out[256… {}

In [258…
```
del d
```

In [260…
```
d
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[260], line 1
----> 1 d

NameError: name 'd' is not defined
```

In [264…
```
help()
```

```
Welcome to Python 3.12's help utility! If this is your first time using
Python, you should definitely check out the tutorial at
https://docs.python.org/3.12/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To get a list of available
modules, keywords, symbols, or topics, enter "modules", "keywords",
"symbols", or "topics".

Each module also comes with a one-line summary of what it does; to list
the modules whose name or summary contain a given string such as "spam",
enter "modules spam".

To quit this help utility and return to the interpreter,
enter "q" or "quit".
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return bool(key in self).
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, index, /)
 |      Return self[index].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
```

```
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
|
```

```
|  copy(self, /)
|      Return a shallow copy of the list.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  extend(self, iterable, /)
|      Extend list by appending elements from the iterable.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  insert(self, index, object, /)
|      Insert object before index.
|
|  pop(self, index=-1, /)
|      Remove and return item at index (default last).
|
|      Raises IndexError if list is empty or index is out of range.
|
|  remove(self, value, /)
|      Remove first occurrence of value.
|
|      Raises ValueError if the value is not present.
|
|  reverse(self, /)
|      Reverse *IN PLACE*.
|
|  sort(self, /, *, key=None, reverse=False)
|      Sort the list in ascending order and return None.
|
|      The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
|      order of two equal elements is maintained).
|
|      If a key function is given, apply it once to each list item and sort them,
|      ascending or descending, according to their function values.
|
|      The reverse flag can be set to sort in descending order.
|
|  ----------------------------------------------------------------------
```

```
|  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

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
```

In [266…

```
help(list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return bool(key in self).
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, index, /)
 |      Return self[index].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
```

```
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
|
```

```
|  copy(self, /)
|      Return a shallow copy of the list.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  extend(self, iterable, /)
|      Extend list by appending elements from the iterable.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  insert(self, index, object, /)
|      Insert object before index.
|
|  pop(self, index=-1, /)
|      Remove and return item at index (default last).
|
|      Raises IndexError if list is empty or index is out of range.
|
|  remove(self, value, /)
|      Remove first occurrence of value.
|
|      Raises ValueError if the value is not present.
|
|  reverse(self, /)
|      Reverse *IN PLACE*.
|
|  sort(self, /, *, key=None, reverse=False)
|      Sort the list in ascending order and return None.
|
|      The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
|      order of two equal elements is maintained).
|
|      If a key function is given, apply it once to each list item and sort them,
|      ascending or descending, according to their function values.
|
|      The reverse flag can be set to sort in descending order.
|
|  ----------------------------------------------------------------------
```

```
|  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
```

In [268…
```
bin(15)
```

Out[268…
```
'0b1111'
```

In [270…
```
oct(15)
```

Out[270…
```
'0o17'
```

In [272…
```
0o17
```

Out[272…
```
15
```

In [274…
```
hex(10)
```

Out[274…
```
'0xa'
```

In [276…
```
hex(3)
```

Out[276…
```
'0x3'
```

swap in python

In [279…
```
a = 5
b = 6
```

In [281…
```
print(a)
print(b)
```

```
5
6
```

In [283…
```
a,b = b,a
```

In [285…
```
print(a)
print(b)
```

```
6
5
```

In [287…
```
a = a + b
b = a - b
a = a - b
```

In [289…
```
print(a)
print(b)
```

```
5
6
```

In [291…
```
a = [1,2,3]
b = [1,2,3]
r1 = a == b                    # checks same value or not
r2 = a is b                    # checks same memory location or not
print(r1)
print(r2)
```

```
True
False
```

In [ ]:
```
# short circuiting behaviour by logical 'or' and 'and' operator
```