In [ ]:
```
# set
```

In [73]:
```
s3={}
print(type(s3))
s3=set({})
print(type(s3))
```

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

In [74]:
```
help(int)
```

```
Help on class int in module builtins:

class int(object)
 |  int([x]) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
 |
 |  If x is not a number or if base is given, then x must be a string,
 |  bytes, or bytearray instance representing an integer literal in the
 |  given base.  The literal can be preceded by '+' or '-' and be surrounded
 |  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
 |  Base 0 means to interpret the base from the string as an integer literal.
 |  >>> int('0b100', base=0)
 |  4
 |
 |  Built-in subclasses:
 |      bool
 |
 |  Methods defined here:
 |
 |  __abs__(self, /)
 |      abs(self)
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __bool__(self, /)
 |      self != 0
 |
 |  __ceil__(...)
 |      Ceiling of an Integral returns itself.
 |
 |  __divmod__(self, value, /)
 |      Return divmod(self, value).
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __float__(self, /)
 |      float(self)
 |
 |  __floor__(...)
 |      Flooring an Integral returns itself.
 |
 |  __floordiv__(self, value, /)
 |      Return self//value.
 |
 |  __format__(self, format_spec, /)
 |      Default object formatter.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __index__(self, /)
 |      Return self converted to an integer, if self is suitable for use as an index into a list.
 |
 |  __int__(self, /)
 |      int(self)
 |
 |  __invert__(self, /)
 |      ~self
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __lshift__(self, value, /)
 |      Return self<<value.
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __neg__(self, /)
 |      -self
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __pos__(self, /)
 |      +self
 |
 |  __pow__(self, value, mod=None, /)
 |      Return pow(self, value, mod).
 |
 |  __radd__(self, value, /)
 |      Return value+self.
 |
 |  __rand__(self, value, /)
 |      Return value&self.
 |
 |  __rdivmod__(self, value, /)
 |      Return divmod(value, self).
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rfloordiv__(self, value, /)
 |      Return value//self.
 |
 |  __rlshift__(self, value, /)
 |      Return value<<self.
 |
 |  __rmod__(self, value, /)
 |      Return value%self.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __ror__(self, value, /)
 |      Return value|self.
 |
 |  __round__(...)
 |      Rounding an Integral returns itself.
 |      Rounding with an ndigits argument also returns an integer.
 |
 |  __rpow__(self, value, mod=None, /)
 |      Return pow(value, self, mod).
 |
 |  __rrshift__(self, value, /)
 |      Return value>>self.
 |
 |  __rshift__(self, value, /)
 |      Return self>>value.
 |
 |  __rsub__(self, value, /)
 |      Return value-self.
 |
 |  __rtruediv__(self, value, /)
 |      Return value/self.
 |
 |  __rxor__(self, value, /)
 |      Return value^self.
 |
 |  __sizeof__(self, /)
 |      Returns size in memory, in bytes.
 |
 |  __sub__(self, value, /)
 |      Return self-value.
 |
 |  __truediv__(self, value, /)
 |      Return self/value.
 |
 |  __trunc__(...)
 |      Truncating an Integral returns itself.
 |
 |  __xor__(self, value, /)
 |      Return self^value.
 |
 |  as_integer_ratio(self, /)
 |      Return integer ratio.
 |
 |      Return a pair of integers, whose ratio is exactly equal to the original int
 |      and with a positive denominator.
 |
 |      >>> (10).as_integer_ratio()
 |      (10, 1)
 |      >>> (-10).as_integer_ratio()
 |      (-10, 1)
 |      >>> (0).as_integer_ratio()
 |      (0, 1)
 |
 |  bit_length(self, /)
 |      Number of bits necessary to represent self in binary.
 |
 |      >>> bin(37)
 |      '0b100101'
 |      >>> (37).bit_length()
 |      6
 |
 |  conjugate(...)
 |      Returns self, the complex conjugate of any int.
 |
 |  to_bytes(self, /, length, byteorder, *, signed=False)
 |      Return an array of bytes representing an integer.
 |
 |      length
 |        Length of bytes object to use.  An OverflowError is raised if the
 |        integer is not representable with the given number of bytes.
 |      byteorder
 |        The byte order used to represent the integer.  If byteorder is 'big',
 |        the most significant byte is at the beginning of the byte array.  If
 |        byteorder is 'little', the most significant byte is at the end of the
 |        byte array.  To request the native byte order of the host system, use
 |        `sys.byteorder' as the byte order value.
 |      signed
 |        Determines whether two's complement is used to represent the integer.
 |        If signed is False and a negative integer is given, an OverflowError
 |        is raised.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  from_bytes(bytes, byteorder, *, signed=False) from builtins.type
 |      Return the integer represented by the given array of bytes.
 |
 |      bytes
 |        Holds the array of bytes to convert.  The argument must either
 |        support the buffer protocol or be an iterable object producing bytes.
 |        Bytes and bytearray are examples of built-in objects that support the
 |        buffer protocol.
 |      byteorder
 |        The byte order used to represent the integer.  If byteorder is 'big',
 |        the most significant byte is at the beginning of the byte array.  If
 |        byteorder is 'little', the most significant byte is at the end of the
 |        byte array.  To request the native byte order of the host system, use
 |        `sys.byteorder' as the byte order value.
 |      signed
 |        Indicates whether two's complement is used to represent the integer.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  denominator
 |      the denominator of a rational number in lowest terms
 |
 |  imag
 |      the imaginary part of a complex number
 |
 |  numerator
 |      the numerator of a rational number in lowest terms
 |
 |  real
 |      the real part of a complex number
```

In [75]:
```
st5={34,"Hello",34,45,(43,4,5)}
st5
```

Out[75]:
```
{(43, 4, 5), 34, 45, 'Hello'}
```

In [76]:
```
st2={"A","B",34,"Hello"}
st2
```

Out[76]:
```
{34, 'A', 'B', 'Hello'}
```

In [77]:
```
st3=st1.union(st2)
st3
```

Out[77]:
```
{(43, 4, 5), 34, 45, 'A', 'B', 'Hello'}
```

In [78]:
```
st2.add("D")
st2.add("A")
st2
```

Out[78]:
```
{34, 'A', 'B', 'D', 'Hello'}
```

In [79]:
```
st3={5,6,7}
st4={6,7,8}
st3.update(st4)
st3
```

Out[79]:
```
{5, 6, 7, 8}
```

In [80]:
```
st4=st3.intersection(st4)
st4
```

Out[80]:
```
{6, 7, 8}
```

In [81]:
```
st4.issubset(st3)
```

Out[81]:
```
True
```

In [82]:
```
st3.issuperset(st4)
```

Out[82]:
```
True
```

In [83]:
```
# Dictnio
st5={4,8,9}
st6={1,2,3}
st5.isdisjoint(st6)
```

Out[83]:
```
True
```

In [84]:
```
st5.remove(4)
st5
```

Out[84]:
```
{8, 9}
```

In [85]:
```
st6.remove(2)
st6
```

Out[85]:
```
{1, 3}
```

In [86]:
```
st6.clear()
st6
```

Out[86]:
```
set()
```

In [87]:
```
st5==st6
```

Out[87]:
```
False
```

In [88]:
```
st6=(4,9,8)
st5==st5
```

Out[88]:
```
True
```

In [89]:
```
min(st6)
```

Out[89]:
```
4
```

In [90]:
```
max(st6)
```

Out[90]:
```
9
```

In [91]:
```
len(st6)
```

Out[91]:
```
3
```

In [92]:
```
sum(st6)
```

Out[92]:
```
21
```

In [93]:
```
# Frozenset
fs=frozenset()
```

In [109]:
```
fs
```

Out[109]:
```
frozenset()
```

In [94]:
```
type(fs)
```

Out[94]:
```
frozenset
```

In [95]:
```
namelist1=["Kirti","Trupti","Nagma","Poornima"]
fs1=frozenset(namelist1)
fs1
```

Out[95]:
```
frozenset({'Kirti', 'Nagma', 'Poornima', 'Trupti'})
```

In [96]:
```
namelist2=["srushti","sudaxi","Trupti","megha","kajal"]
fs2=frozenset(namelist2)
fs2
```

Out[96]:
```
frozenset({'Trupti', 'kajal', 'megha', 'srushti', 'sudaxi'})
```

In [97]:
```
fs2.union(fs1)
```

Out[97]:
```
frozenset({'Kirti',
           'Nagma',
           'Poornima',
           'Trupti',
           'kajal',
           'megha',
           'srushti',
           'sudaxi'})
```

In [98]:
```
fs2.intersection(fs1)
```

Out[98]:
```
frozenset({'Trupti'})
```

In [99]:
```
fs2.isdisjoint(fs1)
```

Out[99]:
```
False
```

In [100]:
```
fs2.issubset(fs1)
```

Out[100]:
```
False
```

In [101]:
```
fs2.issuperset(fs1)
```

Out[101]:
```
False
```

In [102]:
```
fs1=frozenset("Hello")
fs1
```

Out[102]:
```
frozenset({'H', 'e', 'l', 'o'})
```

In [ ]:
```

```