In [ ]:

```
class Rect:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    @property
    def area(self):
        return self.x * self.y


class Square(Rect):
    def __init__(self, z):
        super().__init__(x=z,y=z)

r = Rect(2, 3)
print(r.area)                        #   6
s = Square(5)
print(s.area)                        #   25
```

In [ ]:

```
class A:
     def __init__(self):
         self.__x = 1
         self.y = 2

     def p(self):
         print(self.__x, self.y)

class B(A):
     def __init__(self):
         super().__init__()
         self.__x = 3
         self.y = 4

b = B()
b.p()                                   #  1    4
print(b._B__x)                          #  3
```

In [ ]:

```
class ClipBoard:
    def __init__(self, target):
        self.target = target
        self.__message = None

    def fill(self, text):
        self.__message = text

    def clear(self):
        self.__message = None

class ExtendedClipBoard(ClipBoard):
    def __init__(self, target, message):
        ClipBoard.target = target
        ClipBoard.message = message
        self.store = None

    def save(self):
        self.store = ClipBoard.target + ClipBoard.message

    def remove(self):
        self.store = None

ob = ClipBoard("ali")
print(ob.target)                             # ali
ob.fill('farshid')
print(ob._ClipBoard__message)                # farshid

obj = ExtendedClipBoard("ali", "taha")
obj.save()
print(obj.store)                             # alitaha
obj.remove()
print(obj.store)                             # None
```

In [ ]:

```
class A:
    def __init__(self):
        print('A')
        self.x = 5

    def func(self):
        self.x = 2

class B(A):
    def func(self):
        self.x += 1
        return self.x

b = B()                                         # A
print(b.func())                                 # 6
```

In [ ]:

```python
class A:
    def __str__(self):
        return "A"

class B(A):
    def __str__(self):
        return "B"

class C(B):
    pass

ob = C()
print(ob)                                    # B
```

In [ ]:

```python
class A:
    def __init__(self):
        print('A')
        super().__init__()

    def __str__(self):
        return "hello"

class B(A):
    def __init__(self):
        print('B')

class C(B):
    def __init__(self):
        print('C')
        super().__init__()

b = B()                                    # B
print(b)                                   # hello
c = C()                                    # C  B
print(c)                                   # hello
```

In [ ]:

```python
class A:
    def h(self):
        return "A"

    def f(self):
        print(self.h())

class B(A):
    def h(self):
        return "B"

A().f()                                    # A
B().f()                                    # B
```

In [ ]:

```
class A:
   def f(self):
      print('1')

class B(A):
   def f(self):
      print('2')
      super().f()

class C(B):
   def f(self):
      print('3')
      super().f()

obj = C()
obj.f()                                    # 3 2  1
```

In [ ]:

```
class A:
    def __init__(self):
        self.f(4)
        print(self.x)

    def f(self, x):
        self.x = 3 * x;

class B(A):
    def __init__(self):
        super().__init__()

b = B()                                      # 12
```

In [ ]:

```
class A:
    def __init__(self):
        self.f(4)
        print(self.x)

    def f(self, x):
        self.x = 3 * x;

class B(A):
    def __init__(self):
        super().__init__()

    def f(self, x):
        self.x = 2 * x;

b = B()                                       # 8
```

In [ ]:

```
class B:
    x=0
    def __init__(self):
        x=1
        print("B")

class D(B):
    def __init__(self):
        super().__init__()
        global x
        print(x)
        x=2
        print("D")

ob = D()                                     # B  1  D
print(x)                                     # 2
```

In [ ]:

```
class A:
    def __init__(self, x = 1):
        self.x = x

    def f(self):
        self.x += 2

class B(A):
    def __init__(self, y = 3):
       A.__init__(self, 4)
       self.y = y

    def f(self):
        self.y += 5

def main():
    b = B()
    print(b.x, b.y)                         # 4  3

    b.f()
    print(b.x, b.y)                         # 4  8

main()
```

In [ ]:

```python
class Person:
    def __init__(self, name, job=None, pay=0):
        self.name = name
        self.job = job
        self.pay = pay


    def f(self, percent):
        self.pay = int(self.pay * (1 + percent))

    def __repr__(self):
        return '[Person: %s, %s]' % (self.name, self.pay)


class Manager(Person):
    def __init__(self, name, pay):
        Person.__init__(self, name, 'mgr', pay)

    def f(self, percent, bonus=.10):
        Person.f(self, percent + bonus)

if __name__ == '__main__':
    ali  = Person('Ali')
    sara = Person('Sara', job='dev', pay=10)
    taha = Manager('Taha', 40)

    for i in (ali, sara, taha):
        i.f(.10)
        print(i)
```

In [ ]:

```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, p2):
        return math.sqrt((self.x-p2.x)**2 + (self.y-p2.y)**2)

class Polygon:
    def __init__(self):
        self.vertices = []

    def add_point(self, point):
        self.vertices.append((point))

    def perimeter(self):
        p = 0
        points = self.vertices + [self.vertices[0]]
        for i in range(len(self.vertices)):
            p += points[i].distance(points[i+1])
        return p

square = Polygon()
square.add_point(Point(1,1))
square.add_point(Point(1,2))
square.add_point(Point(2,2))
square.add_point(Point(2,1))
print(square.perimeter())                      # 4.0
```

In [ ]:

```
from math import sqrt

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return sqrt(self.x * self.x + self.y * self.y)

    def distance(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        return sqrt(dx * dx + dy * dy)

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy

    def __str__(self):
        return "(" + str(self.x) + ", " + str(self.y) + ")"


class Point3D(Point):
      z = 0
      def __init__(self, x, y, z):
           Point.__init__(self, x, y)
           self.z = z

      def translate(self, dx, dy, dz):
           Point.translate(self, dx, dy)
           self.z += dz

p = Point(3, -4)
p.translate(1, 5)
print(p)                                    # (4, 1)
Point.translate(p, 1, 5)
print(p)                                    # (5, 6)
q = Point3D(1, 2 , 3)
print(q)                                    # (1, 2)
```

In [ ]:

```
class AudioFile:
    def __init__(self, filename):
        if not filename.endswith(self.ext):
            raise Exception("Invalid file format")
        self.filename = filename

class MP3File(AudioFile):
    ext = "mp3"
    def play(self):
        print("playing {} as mp3".format(self.filename))

class WavFile(AudioFile):
    ext = "wav"
    def play(self):
        print("playing {} as wav".format(self.filename))


mp3 = MP3File("a.mp3")
mp3.play()                    # playing a.mp3 as mp3
```

In [ ]:

```
class atom:
    def __init__(self,atno,x,y,z):
        self.atno = atno
        self.p = (x,y,z)

    def __repr__(self):
        return '%d %10.4f %10.4f %10.4f' %(self.atno, self.p[0],self.p[1],sel

class molecule:
    def __init__(self,name='Generic'):
        self.name = name
        self.atomlist = []

    def addatom(self,atom):
        self.atomlist.append(atom)

    def __repr__(self):
        str = 'This is a molecule named %s\n' % self.name
        str = str+'It has %d atoms\n' % len(self.atomlist)
        for atom in self.atomlist:
            str = str + 'atom' + '\n'
        return str


mol = molecule('Water')
at = atom(8,0.,0.,0.)
print(at)                # 8     0.0000     0.0000     0.0000
mol.addatom(at)
mol.addatom(atom(1,0.,0.,1.))
mol.addatom(atom(1,0.,1.,0.))
print(mol)
'''
This is a molecule named Water
It has 3 atoms
atom
atom
atom
'''
```

In [ ]:

```
class B:
    cb = 0
    def f(self):
        self.cb += 1

class L(B):
    cl = 0
    def f(self):
        B.f(self)
        self.cl += 1

class R(B):
    cr = 0
    def f(self):
        B.f(self)
        self.cr += 1

class S(L, R):
    cs = 0
    def f(self):
        L.f(self)
        R.f(self)
        self.cs += 1

s = S()
s.f()
print(s.cb, s.cl, s.cr, s.cs)               # 2  1  1  1
```

In [ ]:

```
class B:
    a = 0
    def f(self):
        self.a += 1

class L(B):
    b = 0
    def f(self):
        super().f()
        self.b += 1

class R(B):
    c = 0
    def f(self):
        super().f()
        self.c += 1

class S(L, R):
    d = 0
    def f(self):
        super().f()
        self.d += 1

s = S()
s.f()
print(s.a, s.b, s.c, s.d)               # 1  1  1  1
```

In [ ]:

```
class C1:
    def f(self):
        self.__X = 1

    def g(self):
        print(self.__X)

class C2:
    def h(self):
        self.__X = 2

    def w(self):
        print(self.__X)

class C3(C1, C2):
    pass

I = C3()
I.f()
I.h()
print(I.__dict__)               # {'_C1__X': 1, '_C2__X': 2}
I.g()                           # 1
I.w()                           # 2
```

In [ ]:

```
class C:
    def act(self):
        print('C')

class D(C):
    def act(self):
        super().act()
        print('D')

class E(C):
    def m(self):
        p = super()
        print(p)
        p.act()

X = D()
X.act()                       # C  D
print(super)                  # <class 'super'>
E().m()                       # <super: <class 'E'>, <E object>> C
```

In [ ]:

```
class A:
    def act(self):
        print('A')

class B:
    def act(self):
        print('B')

class C(B, A):
    def act(self):
        super().act()

X = C()
X.act()                     # B
```

In [ ]:

```
class B:
    def __init__(self):
        print('B')

class C:
    def __init__(self):
        print('C')

class D(B, C):
    pass

d = D()                     # B
```

In [ ]:

```
class A:
    def __init__(self):
        print('A')

class B(A):
    def __init__(self):
        print('B');
        A.__init__(self)

class C(A):
    def __init__(self):
        print('C');
        A.__init__(self)

x = B()                              # B  A
x = C()                              # C  A
```

In [ ]:

```
class A:
    x = 1

class B(A):
    x = 2

class C(A):
    x = 3

class D(C, B):
    pass

d = D()
print(d.x)                       # 3
```

In [ ]:

```
class A:
    x = 1

class B(A):
    pass

class C(A):
    x = 3

class D(B, C):
    pass

d = D()
print(d.x)                       # 3
```

In [ ]:

```
class A:
    x = 1

class B(A):
    pass

class C(A):
    pass

class D(B, C):
    pass

d = D()
print(d.x)                             # 1
print([cls.__name__ for cls in D.__mro__])   # ['D', 'B', 'C', 'A', 'object']
print(D.__bases__)
#  (<class '__main__.B'>, <class '__main__.C'>)
```

In [ ]:

```
class D(dict):
    def longest_key(self):
        l = None
        for key in self:
            if not l or len(key) > len(l):
                l = key
        return l

ob = D()
ob['sara'] = 1
ob['farshid'] = 5
ob['taha'] = 3
print(ob)                       #{'sara': 1, 'farshid': 5, 'taha': 3}
print(ob.longest_key())         # farshid
```

In [ ]:

```
class ContactList(list):
    def search(self, name):
        mc = []   # matching_contacts
        for c in self:
            if name in c.name:
                mc.append(c)
        return mc


class Contact:
    ac = ContactList()

    def __init__(self, name, email):
        self.name = name
        self.email = email
        self.ac.append(self)

c1 = Contact("Ali reza" , "ali@gmail.com")
c2 = Contact("Ali taha" , "ali@gmail.com")
c3 = Contact("Sara Z"   , "sara@gmail.com")

print([c.name for c in Contact.ac.search('Ali')])
# ['Ali reza', 'Ali taha']
```

In [ ]:

```
class MyList(list):
    def __getitem__(self, offset):
        return list.__getitem__(self, offset - 1)

if __name__ == '__main__':
    lst = list('abc')
    print(lst[1])                # b

    x = MyList('abc')
    print(x[1])                  # a

    x.append('d')
    x.reverse()
    print(x)                     # ['d', 'c', 'b', 'a']
```

In [ ]:

```
class C:
    def __init__(self, value = []):
        self.data = []
        self.concat(value)

    def intersect(self, other):
        res = []
        for x in self.data:
            if x in other:
                res.append(x)
        return C(res)

    def union(self, other):
        res = self.data[:]
        for x in other:
            if not x in res:
                res.append(x)
        return C(res)

    def concat(self, value):
        for x in value:
            if not x in self.data:
                self.data.append(x)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, key):
        return self.data[key]

    def __and__(self, other):
        return self.intersect(other)

    def __or__(self, other):
        return self.union(other)

    def __repr__(self):
        return repr(self.data)

    def __iter__(self):
        return iter(self.data)



x = C([1, 3, 5])
print(x.union(C([1, 4])))                   # [1, 3, 5, 4]
print(x | C([1, 4]))                        # [1, 3, 5, 4]
```

In [ ]:

```
class MySet(list):
    def __init__(self, value = []):
        list.__init__([])
        self.concat(value)

    def intersect(self, other):
        res = []
        for x in self:
            if x in other:
                res.append(x)
        return MySet(res)

    def union(self, other):
        res = MySet(self)
        res.concat(other)
        return res

    def concat(self, value):
        for x in value:
            if not x in self:
                self.append(x)

    def __and__(self, other):
        return self.intersect(other)

    def __or__(self, other):
        return self.union(other)

    def __repr__(self):
        return 'Set:' + list.__repr__(self)

if __name__ == '__main__':
    x = MySet([1, 3, 5])
    y = MySet([3, 6])
    print(len(x))                            # 3

    x.reverse()
    print(x)                                 #Set:[5, 3, 1]

    print(x.intersect(y))                    # Set:[3]
    print(x & y)                             # Set:[3]

    print(y.union(x))                        # Set:[5, 3, 1, 6]
    print(x | y)                             # Set:[5, 3, 1, 6]
```

In [ ]:

```
def classtree(cls, i):                    # i : indent
    print('.' * i + cls.__name__)
    for sc in cls.__bases__:
        classtree(sc, i+2)

def instancetree(inst):
    classtree(inst.__class__, 1)

def test():
    class A:
        pass

    class B(A):
        pass

    class C(A):
        pass

    class D(B,C):
        pass

    class E:
        pass

    class F(D,E):
        pass

    instancetree(F())


if __name__ == '__main__':
    test()
```

دانشگاه شهید مدنی آذربایجان
برنامه نویسی پیشرفته با پایتون
امین گلزاری اسکوئی
١٤٠٠-١٤٠١

[Codes and Projects (click here) (https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Advanced-2021)](https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Advanced-2021) [slides and videos (click here) (https://drive.google.com/drive/folders/1Dx3v7fD1QBWL-MNP2hd7iIxaRbeALkkA)](https://drive.google.com/drive/folders/1Dx3v7fD1QBWL-MNP2hd7iIxaRbeALkkA)