In [ ]:
```
class Life:
    def __init__(self, name='unknown'):
        print('Hello ' + name)
        self.name = name

    def live(self):
        print(self.name)

    def __del__(self):
        print('Goodbye ' + self.name)

ob = Life('Sara')                       # Hello Sara
ob.live()                               # Sara
ob = 'Ali'                              # Goodbye Sara
```

In [ ]:
```
class C:
    def __init__(self, a):
        self.a = a

    def f(self, x, y):
        return( self.a + x + y)

ob = C(1)
print(ob.f(2, 3))                       # 1+2+3 =6
print(C.f(ob, 2, 3))                    # 6
```

In [ ]:
```
def add(obj, k):
    obj.t += 1
    k += 1

class A:
    def __init__(self):
        self.t = 1

def main():
    ob = A()
    k = 0
    add(ob,k)
    add(ob,k)
    print(ob.t)
    print(k)

main()                                  #3  0
```

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

    def dist(self, pt):
        a = pt.x - self.x
        b = pt.y - self.y
        return math.sqrt(a ** 2 + b ** 2)

p1 = Point(2, 3)
p2 = Point(3, 3)
print(p1.dist(p2) )                          # 1.0
```

In [ ]:
```
class Person:
    def __init__(self, id):
        self.id = id

ali = Person(100)
print (ali.__dict__)                         # {'id':100}
ali.__dict__['age'] = 35
print (ali.__dict__)                         # {'id':100 ,'age':35}
print (len(ali.__dict__))                    # 2
```

In [ ]:
```
class B:
    def __init__(self,a,b,c):
        self.a = a
        self._b = b
        self.__c = c

    def f(self):
        print(self.a)
        print(self._b)
        print(self.__c)

ob = B(1, 2, 3)
print(ob.a)                                  # 1
print(ob._b)                                 # 2
print(ob._B__c)                              # 3
```

In [ ]:
```
def formatting(lowerscase=False):
     def d(func):
         def w(text=''):
             if lowerscase:
                 func(text.lower())
             else:
                 func(text.upper())
         return w
     return d

@formatting(lowerscase=True)
def f(s):
    print(s)

f("Python")                                    # python
```

In [ ]:
```
class B:
   def __init__(self, s):
     self.s = s

   @classmethod
   def f(cls, lst):
     x = cls('')
     x.s = '-'.join(str(i) for i in lst)
     return x

   def __str__(self):
     return self.s

a = ['5','8','6']
ob = B.f(a)
print(ob)                                      # 5-8-6
```

In [ ]:
```
class Person:
    TITLES = ('Mr', 'Mrs', 'Ms')

    @classmethod
    def f(cls, a):
        return [t for t in cls.TITLES if t.endswith(a)]

ob = Person()
print(ob.f("s"))                               # ['Mrs', 'Ms']
```

In [ ]:
```
class C:
    def f(self, x):
        print([self, x])

    def s(x):
        print(x+3)

    def h(cls, x):
        print([cls, x])

    s = staticmethod(s)
    h = classmethod(h)

obj = C()
obj.f(5)                                # [<__main__.C object at ...>, 5]
C.s(1)                                  # 4
C.h(3)                                  # [<class '__main__.C'>, 3]
```

In [ ]:
```
class C:
    n = 0
    def __init__(self):
        C.n += 1

    def p():
        print(C.n)

a = C()
b = C()
C.p()                                   # 2
```

In [ ]:
```
class C:
    x = 2

ob = C()
k = lambda: ob.x + 3
print(k())                              # 5

print('----')

class C:
    pass

ob = C()
print(ob.__class__)                      # <class '__main__.C'>
print(isinstance(ob, C))                 # True
print(C.__bases__)                       # (<class 'object'>,)
```

In [ ]:
```
class Department:
    def __init__( self ):
        self.lst = []

    def f( self, s ):
       self.lst.append(s)

class Student:
    def __init__( self, name ):
        self.name = name

d = Department()
d.f(Student("Ali"))
d.f(Student("Farshid"))

for s in d.lst:
    print( "%s" % (s.name))                    #Ali  Farshid
```

In [ ]:
```
class C:
    def __init__(self):
        self.a = 5

class E:
    def __init__(self, x, y=None):
        self.x = x
        self.y = y


ob = E(2,C())
print(ob.x)                                    # 2
print(ob.y.a)                                  # 5
```

In [ ]:
```
class C:
     x = 4
     def f(self):
         print("F")

     @property
     def g(self):        # can be called as attribute only
         return("G")

ob = C()
ob.f()                                         # F
print(ob.x)                                    # 4
print(ob.g)                                    # G
#print(ob.g())                                 # Error
```

In [ ]:
```
## An instance can be used as function if the class method
## contains __call__ method.

class C:
     def __init__(self, n=0):
         self.n = n

     def __call__(self, n):
        self.n = n

ob = C()
print(ob.n)                                          # 0
ob(5)
print(ob.n)                                          # 5
```

In [ ]:
```
## Changing Class Attributes Can Have Side Effects
class C:
    a = 1                                    # Class attribute

ob1 = C()
print(ob1.a)                                         # 1

C.a = 2
print(ob1.a)                                         # 2

ob2 = C()
print(ob2.a)                                         # 2
```

In [ ]:
```
##Changing Mutable Class Attributes Can Have Side Effects.
class C:
    s = []                                # Class attribute
    def __init__(self):
        self.p = []                       # Instance attribute

x = C()
y = C()
print(y.s, y.p)                                  #[]  []
x.s.append('a')
x.p.append('a')
print(x.s, x.p)                                  #['a']   ['a']
print(y.s, y.p)                                  #['a']   []
```

In [ ]:
```
def f():
    class C:
        a = 1
        def m(self):
            C.a += 2
            return(C.a)
    return C()

print(f().m())                                   # 3
```

In [ ]:
```
class C:
    a = 1
    def m(self):
        print(C.a)

def f():
    return C()

f().m()                                    # 1
```

In [ ]:
```
def d(func):
    def f(*args):
        f.c += 1
        print(f.c)
        return func(*args)
    f.c = 0
    return f

class C:
    @d
    def g(self,a, b):
        return a + b

ob = C()
print(ob.g(1, 2))                          # 1  3
print(ob.g('ali', 'reza'))                 # 2  alireza
```

In [ ]:
```
class T:
    def __init__(self, func):
        self.c = 0
        self.func = func

    def __call__(self, *args):
        self.c += 1
        print(self.c)
        return self.func(*args)

@T
def g(a, b):
    return(a + b)

print(g(1, 2))                             # 1  3
print(g('ali', 'reza'))                    # 2  alireza
```

In [ ]:
```
# Decorator Nesting
def d1(F):
    return (lambda: 'X' + F())

def d2(F):
    return (lambda: 'Y' + F())

@d1
@d2
def func():                                     # func = d1(d2(func))
    return 's'

print(func())                                   # XYs
```

In [ ]:
```
class Squares:
    def __init__(self, start, stop):
        self.start = start
        self.stop = stop

    def __iter__(self):
        for v in range(self.start, self.stop + 1):
            yield v ** 2

for i in Squares(1, 3):
    print(i)                                    # 1  4  9
```

In [ ]:
```
class C:
    def f(self):
        print("1")
        return self._a

    def g(self, value):
        print("2")
        self._a = value

    def h(self):
        print("3")
        del self._a

    a = property(f, g, h)

ob = C()
ob.a = "sara"                                   # 2
print(ob.a)                                     # 1  sara
del ob.a                                        # 3
```

In [ ]:
```
# Methods Are Objects
class C:
    def f(self, message):
        print(message)

ob = C()
g = ob.f
g('Amin')                                   # Amin

h = C.f
h(ob, 'Amin')                               # Amin
```

In [ ]:
```
class C:
    def f(self, n):
        print(n)

    def g(self):
        x = self.f
        x(5)

C().g()                                         # 5
```

In [ ]:
```
##Classes Are Objects

def f(klass, *pargs, **kargs):
    return klass(*pargs, **kargs)

class C:
   def doit(self, m):
      print(m)

class P:
    def __init__(self, n, j=None):
        self.n = n
        self.j = j

ob = f(C)
ob.doit(1)                                      # 1

y = f(P, 5, "K")
print(y.n, y.j)                                 # 5  K

z = f(P, n=8)
print(z.n, z.j)                                 # 8  None
```

In [ ]:

```
class C:
    def __init__(self, x=2, y=3):
        self.x = x
        self.y = y

    def __str__(self):
        return "A"

    def __eq__(self, o ):
        return self.x * self.y == o.x * o.y

def main():
    a = C(1, 4)
    b = C(2, 2)
    print(a == b)                                   # True

main()
```

In [ ]:

```
class C:
    data = 'b'

    def __gt__(self, other):
        return self.data > other

    def __lt__(self, other):
        return self.data < other

ob = C()
print( ob < 'd')                                    # True
print( ob > 'a')                                    # True
```

In [ ]:

```
class Cursor:
    def __init__(self, doc):
        self.doc = doc
        self.p = 0

    def forward(self):
        self.p += 1

    def back(self):
        self.p -= 1

    def home(self):
        while self.doc.lst[self.p-1] != '\n':
            self.p -= 1
            if self.p == 0:
                break

    def end(self):
        while self.p < len(self.doc.lst) and self.doc.lst[self.p] != '\n':
            self.p += 1

class Document:
    def __init__(self,filename):
        self.lst = []
        self.cursor = Cursor(self)
        self.filename = filename

    def insert(self, character):
        self.lst.insert(self.cursor.p,character)
        self.cursor.forward()

    def delete(self):
        del self.lst[self.cursor.p]

    def save(self):
        f = open(self.filename, 'w')
        f.write(''.join(self.lst))
        f.close()

    @property
    def string(self):
        return "".join(self.lst)


d = Document('a.txt')
d.insert('G')
d.insert('o')
d.insert('l')
d.insert('z')
d.insert('a')
d.insert('r')
d.insert('i')

print(d.string)                                    # Golzari

d.cursor.home()
```

```
d.insert("*")
print(d.string)                                    # *Golzari

d.save()
```

دانشگاه شهید مدنی آذربایجان
برنامه نویسی پیشرفته با پایتون
امین گلزاری اسکوئی
۱۴۰۰-۱۴۰۱

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

In [ ]: