实验一没什么难点，记得四则运算函数返回值前面加Rational把它变成Rational型就好，否则会被视为元组。

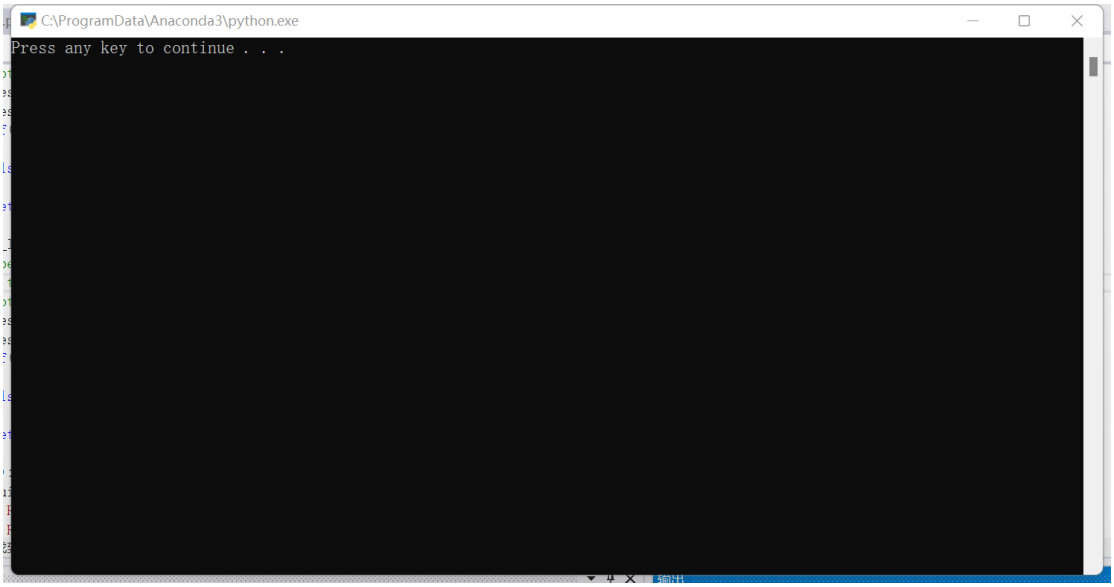
```
import sys
import math
savedStdout=sys.stdout
with open('out.txt','w+') as file:
    stdout=file

def gcd(a, b):
    if a == b:
        return a
    elif a > b:
        return gcd(a-b, b)
    else:
        return gcd(a, b-a)

class Rational:
    def __init__(self,n=0, d=1,):
        # to be implemented
        # e.g. transform 120/-64 to -15/8
        _nu = n; _de = d
        self.__dict__['nu'] = _nu; self.__dict__['de'] = _de


    def __setattr__(self, name, value):
        raise TypeError('Error: Rational objects anu demutable')

    def __str__(self): return '%d/%d' % (self.nu, self.de)

    def __add__(self, other):
    # to be implemented
        #other=Rational(other)
        result_nu=self.nu*other.de+other.nu*self.de
        result_de=self.de*other.de
        divisor=gcd(abs(result_nu),abs(result_de))
        if(result_nu*result_de<0):
            return Rational((-1)*abs(result_nu),abs(result_de))
        else:
            return Rational(abs(resulr_nu),abs(result_de))
```

```
def __sub__(self, other):
# to be implemented
    #other=Rational(other)
    result_nu=self.nu*other.de-other.nu*self.de
    result_de=self.de*other.de
    divisor=gcd(abs(result_nu),abs(result_de))
    if(result_nu*result_de<0):
        return Rational((-1)*abs(result_nu),abs(result_de))
    else:
        return Rational(abs(resulr_nu),abs(result_de))

def __mul__(self, other):
# to be implemented
    #other=Rational(other)
    result_nu=self.nu*other.nu
    result_de=self.de*other.de
    divisor=gcd(abs(result_nu),abs(result_de))
    if(result_nu*result_de<0):
        return Rational((-1)*abs(result_nu),abs(result_de))
    else:
        return Rational(abs(result_nu),abs(result_de))

def __truediv__(self, other):
# to be implemented
    #other=Rational(other)
    result_nu=self.nu*other.de
    result_de=self.de*other.nu
    divisor=gcd(abs(result_nu),abs(result_de))
    if(result_nu*result_de<0):
        return Rational((-1)*abs(result_nu),abs(result_de))
    else:
        return Rational(abs(resulr_nu),abs(result_de))
def __eq__(self, other):
# to be implemented
#equal
    #other=Rational(other)
    #if(other.nu==self.de):
    if(self.nu*other.de==other.nu*self.de):
        result_bool=True
    else:
        result_bool=False
    return result_bool

def __ne__(self, other):
```

```
# to be implemented
#not equal to
    #other=Rational(other)
    return not(self==other)

def __gt__(self, other):
# to be implemented
#greater than
    #other=Rational(other)
    result_self_judge=self.nu*self.de*other.de*other.de
    result_other_judge=other.nu*other.de*self.de*self.de
    if(result_self_judge>result_other_judge):
        result_bool=True
    else:
        result_bool=False
    return result_bool

def __lt__(self, other):
# to be implemented
#less than
    #other=Rational(other)
    result_self_judge=self.nu*self.de*other.de*other.de
    result_other_judge=other.nu*other.de*self.de*self.de
    if(result_self_judge<result_other_judge):
        result_bool=True
    else:
        result_bool=False
    return result_bool

def __ge__(self, other):
# to be implemented
#greater than or equal to
    #other=Rational(other)
    result_self_judge=self.nu*self.de*other.de*other.de
    result_other_judge=other.nu*other.de*self.de*self.de
    if(result_self_judge>=result_other_judge):
        result_bool=True
    else:
        result_bool=False
    return result_bool

def __le__(self, other):
# to be implemented
#less than or equal to
```

```
        #other=Rational(other)
        result_self_judge=self.nu*self.de*other.de*other.de
        result_other_judge=other.nu*other.de*self.de*self.de
        if(result_self_judge<=result_other_judge):
            result_bool=True
        else:
            result_bool=False
        return result_bool

def test():
    testsuite = [
        ('Rational(2, 3) + Rational(-70, 40)', Rational(-13, 12)),
        ('Rational(-20, 3) - Rational(120, 470)', Rational(-976, 141)),
        ('Rational(-6, 19) * Rational(-114, 18)', Rational(2, 1)),
        ('Rational(-6, 19) / Rational(-114, -28)', Rational(-28, 361)),

        ('Rational(-6, 19) == Rational(-14, 41)', False),
        ('Rational(-6, 19) != Rational(-14, 41)', True),
        ('Rational(6, -19) > Rational(14, -41)', True),
        ('Rational(-6, 19) < Rational(-14, 41)', False),
        ('Rational(-6, 19) >= Rational(-14, 41)', True),
        ('Rational(6, -19) <= Rational(14, -41)', False),
        ('Rational(-15, 8) == Rational(120, -64)', True),
    ]
    for t in testsuite:
        try:
            result = eval(t[0])
        except:
            print ('Error in evaluating ' + t[0]); continue

        if result != t[1]:
            print ('Error:  %s != %s' % (t[0], t[1]))

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

没有报错就好




本实验即写三个函数更改Integrator在子类中的compute_points方法，使之获得点集和权集；再在子函数继承的integrate方法中把这些和线性相加即可。
实验二的第三个函数是错的
首先，第一个问题在于“若n是偶数，则n=n+1”然后在式子中，i的取值从0到n。换句话说，n=4，那么n=5，然后i的取值从0到5共六个值。由于w是一样的，所以提出w，式子中的√3/6项自然就被消掉了。这一项从一开始就没有意义，所以要把条件改成“若n是奇数，则n=n+1”。此时h的系数过大，且每次最终值差2倍，因此考虑去掉h的系数2.此时结果还不够，再考虑把偶数项系数改成1/9，得到刚好到结果。
另一种改法是直接去掉h的系数，但是这样会使得√3/6项无意义。

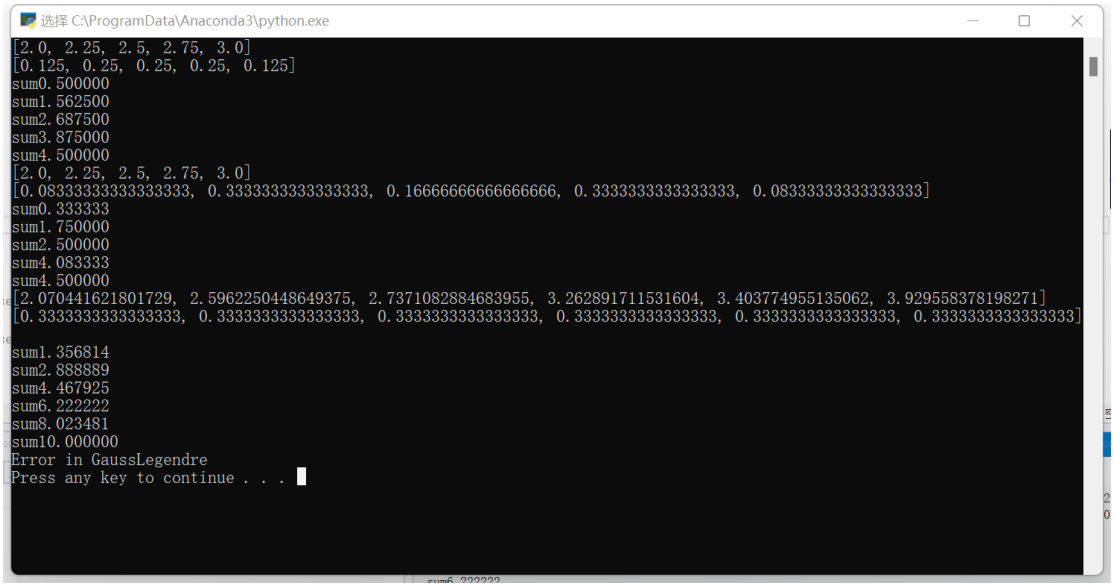
```
import numpy as np
import math

class Integrator(object):
    def __init__(self, a, b, n):
        self.a, self.b, self.n = a, b, n
        self.points, self.weights = self.compute_points()

    def compute_points(self):
        raise NotImplementedError('no rule in class %s' \
                                  % self.__class__.__name__)

    def integrate(self, f):
    # to be implemented
        lenth=len(self.points)
        sum=0
```

```
        for i in range (0,lenth):
            sum=sum+f(self.points[i])*self.weights[i]
            #print(i)
            print('sum%f'%sum)
        return sum


class Trapezoidal(Integrator):
# to be implemented
    def compute_points(self):
        result_point_list=list("")
        result_weight_list=list("")
        h=(self.b-self.a)/self.n
        for i in range (0,self.n+1):
            result_point_list.append(self.a+i*h)
            if(i==0 or i==self.n):
                result_weight_list.append(h/2)
            else:
                result_weight_list.append(h)
        print(result_point_list)
        print(result_weight_list)
        self.points=result_point_list
        self.weight=result_weight_list
        return self.points,self.weight


class Simpson(Integrator):
# to be implemented
    def compute_points(self):
        result_point_list=list("")
        result_weight_list=list("")
        if((self.n)%2==1):
            self.n=self.n+1
        h=(self.b-self.a)/self.n
        for i in range (0,self.n+1):
            result_point_list.append(self.a+i*h)
            if(i==0 or i==self.n):
                result_weight_list.append(h/3)
            elif (i%2==0):
                result_weight_list.append(2*h/3)
            else:
                result_weight_list.append(4*h/3)
        print(result_point_list)
        print(result_weight_list)
```

```
        self.points=result_point_list
        self.weight=result_weight_list
        return self.points,self.weight


class GaussLegendre(Integrator):
# to be implemented
    def compute_points(self):
        result_point_list=list("")
        result_weight_list=list("")
        if((self.n)%2==0):
            self.n=self.n+1
        h=2*(self.b-self.a)/(self.n+1)
        for i in range (0,self.n+1):
            result_weight_list.append(h)
            if (i%2==0):
                result_point_list.append(self.a+(i+1/2-(math.sqrt(3))/6)*h)
            else:
                result_point_list.append(self.a+(i+1/2+(math.sqrt(3))/6)*h)
        print(result_point_list)
        print(result_weight_list)
        self.points=result_point_list
        self.weight=result_weight_list
        return self.points,self.weight


# A linear function will be exactly integrated by all
# the methods, so such an f is the candidate for testing
# the implementations

def test_Integrate():
    """Check that linear functions are integrated exactly."""
    def f(x): return x + 2
    def F(x): return 0.5*x**2 + 2*x

    a = 2; b = 3; n = 4     # test data
    I_exact = F(b) - F(a)
    tol = 1E-6

    methods = [Trapezoidal, Simpson, GaussLegendre]
    for method in methods:
        integrator = method(a, b, n)

        I = integrator.integrate(f)
```

```
        if abs(I_exact - I)/I_exact > tol:
            print ('Error in %s' % method.__name__)

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

输出：
点集
权集
每次加和




更改以后的函数：

```
class GaussLegendre(Integrator):
# to be implemented
    def compute_points(self):
        result_point_list=list("")
        result_weight_list=list("")
        if((self.n)%2==1):
            self.n=self.n+1
        h=(self.b-self.a)/(self.n+1)
        for i in range (0,self.n+1):
            result_weight_list.append(h)
            if (i%2==0):
                result_point_list.append(self.a+(i+1/2-(math.sqrt(3))/9)*h)
            else:
                result_point_list.append(self.a+(i+1/2+(math.sqrt(3))/6)*h)
        print(result_point_list)
        print(result_weight_list)
        self.points=result_point_list
        self.weight=result_weight_list
```

```
        return self.points,self.weight
```

结果：