Open Source SW

"Homework from 2-Basics to 8-Sets&Dicts"

담당교수	황두성 교수님
학번	32164959
이름	허전진

2. Basics

Problem 1

In [20]:

```
n,m = 17, 18
```

In [14]:

```
n // 10 + n % 10
```

Out[14]:

8

In [15]:

```
n % 2 + m % 2
```

Out[15]:

1

In [16]:

```
(m+n) // 2
```

Out[16]:

17

In [17]:

```
(m+n)/2.0
```

Out[17]:

17.5

In [18]:

```
int(0.5*(m+n))
```

Out[18]:

17

In [19]:

```
int(round(0.5*(m+n)))
```

Out[19]:

18

Problem 2

In [24]:

```
s,t="Hello","World"
```

In [25]:

```
len(s)+len(t)
```

Out[25]:

```
10
```

In [27]:

```
s[1] + s[2]
```

Out[27]:

```
'el'
```

In [29]:

```
s[len(s)//2]
```

Out[29]:

```
'l'
```

In [30]:

```
s + t
```

Out[30]:

```
'HelloWorld'
```

In [31]:

```
t + s
```

Out[31]:

```
'WorldHello'
```

In [32]:

```
s * 2
```

Out[32]:

```
'HelloHello'
```

Problem 3-2

In [43]:

```
def func(t):
    return v0*(-0.5)*g*t
```

Problem 3-4

In [45]:

```
v0 = 100
g = 32
t = 3.2
d = 50
```

In [47]:

```
h = - 0.5 * g * t ** 2 + v0 * t + d
```

In [48]:

```
print(h, ' feet')
```

```
206.15999999999997  feet
```

3. Data Types

Problem 1

In [9]:

```
a=-11
b=11
c=9.0
d=b/a
e=c/a
s= 'b/a = %g' % (b/a)
```

In [10]:

```
print(d, "\t", e, "\t", s)
```

```
-1.0    -0.8181818181818182    b/a = -1
```

Problem 2

In [11]:

```
a=3
b=float(a)
c=3.9
d=int(c)
e=round(c)
f=int(round(c))
d=str(c)
e='-4.2'
f=float(e)
```

In [25]:

```
print(type(a),"->",a)
print(type(b),"->",b)
print(type(c),"->",c)
print(type(d),"->",d)
print(type(e),"->",e)
print(type(f),"->",f)
```

```
<class 'int'> -> 3
<class 'float'> -> 3.0
<class 'float'> -> 3.9
<class 'str'> -> 3.9
<class 'str'> -> -4.2
<class 'float'> -> -4.2
```

Problem 4

In [27]:

```
import math as m
```

In [30]:

```
def eq1(x):
    return m.sinh(x)
```

In [52]:

```
def eq2(x):
    return 0.5 * (m.pow(m.e, x) - m.pow(m.e, -x))
```

In [56]:

```
eq1(1) == eq2(1)
```

Out[56]:

```
True
```

Problem 5

In [60]:

```
def y(x):
    return  x*m.tan(c) - (g*x**2) / (2*v0*m.cos(c)**2) + y0
```

In [61]:

```
g = 9.81
c = 1
y0 = 1
v0 = 1
```

In [62]:

```
y(1)
```

Out[62]:

```
-14.244762091441489
```

Problem 6

In [65]:

```
def func(a, p, n):
    return a * (1 + p/100) ** n
```

In [68]:

```
func(10000000,0.05,3)
```

Out[68]:

```
10015007.501249997
```

4. Modules – 1

4. Module(Geometry) - Area.py

In [ ]:

```
def square(s):
    """get A of square with s"""
    A = s**2
    return A


def rectangle(a,b):
    """get A of rectangle with s"""
    A = a*b
    return A

from math import pi

def circle(r):
    """get A of circle with r"""
    A = pi * r**2
    return A


def triangle(b,h):
    """get A of triangle with b, h"""
    A = 0.5*b*h
    return A


def parallelogram(b,h):
    """get A of parallelogram with b,h"""
    A = b*h
    return A

def circular_sector(r,c):
    """get A of circular_sector with r,c"""
    A = pi * (r**2) * (c/360)
    return A


from math import pi

def circle_ring(R,r):
    """get A of circle_ring with R,r"""
    A = pi * (R**2 - r**2)
    return A


def trapezoid(h,a,b):
    """get A of trapezoid with h,a,b"""
    A = h * (a+b) / 2
    return A


def rectangular_box(a,b,c):
    """get A of rectangular_box with a,b,c"""
    A = 2*a*b + 2*b*c + 2*a*c
    return A


def cube(l):
    """get A of cube with l"""
    A=6 * (l**2)
    return A


from math import pi

def cylinder(r,h):
    """get A of cylinder with r,h"""
    A = 2 * pi * r * (r+h)
    return A
```

```
    return r


from math import pi

def right_circular_cone(r,s):
    """get A of right_circular_cone with r,s"""
    A = pi * (r**2) + math.pi * r * s
    return A


from math import pi

def sphere(r):
    """get S of sphere with r"""
    S = 4 * pi * (r**2)
    return S
```

4. Module(Geometry) - busbar.py

In [ ]:

```
from math import sqrt

def right_circular_cone(r,h):
    """get s of right_circular_cone with r,h"""
    s = sqrt((r**2) + (h**2))
    return s
```

4. Module(Geometry) - perimeter.py

In [ ]:

```
def square(s):
    '''get P of square with s'''
    return 4*s


def parallelogram(a, b):
    '''get P of parrallelogram with a, b'''
    return 2*a + 2*b


from math import pi

def circle(r):
    '''get P of circle with r'''
    return 2*pi*r


def triangle(a, b, c):
    '''get P of triangle with a,b,c'''
    return a+b+c


def rectangle(a,b):
    '''get P of rentangle with a,b'''
    return 2*(a+b)


def trapezoid(a, b, c, d):
    '''get P of trapezoid with a,b,c,d'''
    return a+b+c+d;


def circular_sector(r, seta):
    '''get P(length) of circular sector with r, seta'''
    return r * seta
```

4. Module(Geometry) - pythagorean.py

In [ ]:

```
from math import sqrt

def pythagorean_theorem(a,b):
    """get c of pythagorean_theorem with a,b"""
    c = sqrt((a**2) + (b**2))
    return c
```

4. Module(Geometry) - volume.py

In [ ]:

```
from math import pi

def sphere(r):
    '''get volume of sphere with r'''
    return 4 * pi * r ** 3 / 3


def rectangular_box(a,b,c):
    '''get volume of rectangular_box with r'''
    return a * b * c


def right_circular_cone(r, h):
    '''get volume of right_circular_cone with r, h'''
    return (1/3) * pi * r ** 2 * h


def cube(l):
    '''get volume of cube with l'''
    return l ** 3


def cylinder(r, h):
    '''get volume of cylinder with r, h'''
    return pi * r ** 2 * h


def frustum_of_a_cone(r, R, h):
    '''get volume of frustum of a cone with r, R, h'''
    return (1/3) * pi * h * (r**2 + r*R + R**2)
```

4. Module - geometry

In [34]:

```
import nose

def test_module():

    print("***** Geometry Calculator *****\n",'### menu ###\n', 'p - perimeter\n',
          'a - area\n', 'v - volume\n', 'b - busbar\n', 'pytha - pythagorean theorem\n')

    user_menu = str(input())

    if user_menu == 'p':

        import perimeter as p

        print("### perimeter - shape ###\nYou can type\nsquare, rectangle, circle, triangle,
parrelleogram, circular sector, trapezoid\n")

        shape = str(input())

        if shape == 'square':

            print("type - s")
            s = int(input())
            print(p.square(s))

        elif shape == 'rectangle':

            print("type - a, b")
            a = int(input())
            b = int(input())
            print(p.rectangle(a,b))

        elif shape == 'circle':

            print("type - r")
            r = int(input())
            print(p.circle(r))

        elif shape == 'triangle':

            print("type - a, b, c")
            a = int(input())
            b = int(input())
            c = int(input())
            print(p.triangle(a,b,c))

        elif shape == 'parallelogram':

            print("type - a, b")
            a = int(input())
            b = int(input())
            print(p.parallelogram(a,b))

        elif shape == 'circular sector':

            print("type - r, seta")
            r = int(input())
            seta = int(input())
            print(p.circular_sector(r, seta))

        elif shape == 'trapezoid':

            print("type - a, b, c, d")
            a = int(input())
            b = int(input())
            c = int(input())
            d = int(input())
            print(p.trapezoid(a,b,c,d))
```

```
    elif user_menu == 'a':

        import area as ar

        print("### area - shape ###\nYou can type\nsquare, rectangle, circle, triangle,
parallelogram, circular sector, circular ring, trapezoid, rectangular box, right circular cone, cu
be, cylinder\n")

        shape = str(input())

        if shape == 'square':

            print("type - s")
            s = int(input())
            print(ar.square(s))

        elif shape == 'rectangle':

            print("type - a, b")
            a = int(input())
            b = int(input())
            print(ar.rectangle(a,b))

        elif shape == 'circle':

            print("type - r")
            r = int(input())
            print(ar.circle(r))

        elif shape == 'triangle':

            print("type - b, h")
            b = int(input())
            h = int(input())
            print(ar.triangle(b, h))

        elif shape == 'parallelogram':

            print("type - b, h")
            b = int(input())
            h = int(input())
            print(ar.parallelogram(b, h))

        elif shape == 'circular sector':

            print("type - r, seta")
            r = int(input())
            seta = int(input())
            print(ar.circular_sector(r, seta))

        elif shape == 'circular ring':

            print("type - R, r")
            R = int(input())
            r = int(input())
            print(ar.circular_ring(R, r))

        elif shape == 'trapezoid':

            print("type - h, a, b")
            h = int(input())
            a = int(input())
            b = int(input())
            print(ar.trapezoid(h,a,b))

        elif shape == 'rectangular box':

            print("type - a, b, c")
            a = int(input())
            b = int(input())
            c = int(input())
            print(ar.rectangular_box(a, b, c))

        elif shape == 'right circular cone':

            print("type - r, s")
            r = int(input())
```

```
            s = int(input())
            print(ar.right_circular_cone(r,s))

        elif shape == 'cube':

            print("type - l")
            l = int(input())
            print(ar.cube(l))

        elif shape == 'cylinder':

            print("type - r, h")
            r = int(input())
            h = int(input())
            print(ar.cylinder(r,h))

    elif user_menu == 'v':

        import volume as v

        print("### volume - shape ###\nYou can type\nsphere, rectangular box, right circular cone,
cube, cylinder, frustum of a cone\n")

        shape = str(input())

        if shape == 'sphere':

            print("type - r")
            r = int(input())
            print(v.sphere(r))

        elif shape == 'rectangular box':

            print("type - a, b, c")
            a = int(input())
            b = int(input())
            c = int(input())
            print(v.rectangular_box(a,b,c))

        elif shape == 'right circular cone':

            print("type - r, h")
            r = int(input())
            h = int(input())
            print(v.right_circular_cone(r,h))

        elif shape == 'cube':

            print("type - l")
            l = int(input())
            print(v.cube(l))

        elif shape == 'cylinder':

            print("type - r, h")
            r = int(input())
            h = int(input())
            print(v.cyliner(r,h))

        elif shape == 'frustum of a cone':

            print("type - r, R, h")
            r = int(input())
            R = int(input())
            h = int(input())
            print(v.frustum_of_a_cone(r,R,h))

    elif user_menu == 'pytha':

        import pythagorean as pytha

        print("### pythagorean - shape ###\nYou can type\npythagorean theorem\n")

        shape = str(input())

        if shape == 'pythagorean theorem':
```

```
            print("type - a, b")
            a = int(input())
            b = int(input())
            print(pytha.pythagorean_theorem(a,b))

    elif user_menu == 'b':

        import busbar as bb

        print("### busbar - shape ###\nYou can type\nright circular cone\n")

        shape = str(input())

        if shape == 'right circular cone':

            print("type - r, h")
            r = int(input())
            h = int(input())
            print(bb.right_circular_cone(r,h))

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

```
***** Geometry Calulator *****
 ### menu ###
 p - perimeter
 a - area
 v - volume
 b - busbar
 pytha - pythagorean theorem

v
### volume - shape ###
You can type
sphere, rectangular box, right circular cone, cube, cylinder, frustum of a cone

right circular cone
type - r, h
1
3
3.141592653589793
```

4. Modules – 2

Problem 2 - vector distance

In [139]:

```
import numpy as np

u = np.random.randint(10)
v = np.random.randint(10)

vector1 = np.array([u,v]) # vector => u, v

u = np.random.randint(10)
v = np.random.randint(10)

vector2 = np.array([u,v]) # vector => u, v

print(vector1, vector2)
```

```
[9 7] [9 1]
```

Define functions

In [140]:

```
def dist_euclid(v1, v2):
    '''get Euclidean distance'''
    return np.sqrt(np.sum((v1-v2)**2))
```

In [141]:

```
def dist_cityblock(v1, v2):
    '''get Manhattan distance'''
    return np.sum(np.abs(v1-v2))
```

In [142]:

```
def dist_hamming(v1, v2):
    '''get Hamming distance'''
    dist = 0
    for i in range(len(v1)):
        if(v1[i] != v2[i]):
            dist += 1
    return dist
```

In [143]:

```
from numpy import dot
from numpy.linalg import norm

def dist_cosin(v1, v2):
    '''get Cosin distance'''
    return dot(v1, v2) / (norm(v1) * norm(v2))
```

In [144]:

```
def dist_jaccard(v1, v2):
    '''get Tanimoto distance'''
    union = set(v1).union(set(v2))
    intersection = set(v1).intersection(set(v2))
    return len(intersection) / len(union)
```

Test

In [145]:

```
print(dist_euclid(vector1, vector2))
print(dist_cityblock(vector1, vector2))
print(dist_hamming(vector1, vector2))
print(dist_cosin(vector1, vector2))
print(dist_jaccard(vector1, vector2))
```

```
6.0
6
1
0.8523227286486657
0.3333333333333333
```

5. Lists

Problem 1

Get polygon's area

In [1]:

```
import numpy as np

def PolyArea(x,y):
    xy_ = np.dot(x, np.roll(y,1)) - np.dot(y,np.roll(x,1))
    area = 0.5 * np.abs(xy_)
    return area
```

In [3]:

```
x = np.arange(0, 1+0.01, 0.01)
y = np.sqrt(1 - x**2)

print(PolyArea(x,y))
```

```
0.28510425794475935
```

Problem 2

get points of f(x) = x, g(x) = x^2

In [5]:

```
import numpy as np

def f(x):
    return x

def g(x):
    return x**2
```

In [22]:

```
N = int(input("N: "))
E = float(input("E: "))

interval = 8 / N # -4 ~ 4를 N 구간으로 divide
x = np.arange(-4, 4+interval, interval) # x의 범위: -4~4
```

```
N: 400
E: 0.01
```

In [45]:

```
point = [] # E 범위에 만족하는 point가 저장될 list

def CalcPoint(x):
    for _ in x:
        val = f(_) - g(_)
        if(np.abs(val) < E):
            point.append(_)
    return
```

In [46]:

```
CalcPoint(x)
print("오차에 만족하는 근삿값:",point)
```

오차에 만족하는 근삿값: [3.552713678800501e-15, 1.0000000000000044]

Problem 3

PI Scheme - Leibniz vs Euler

In [4]:

```
def LeibnizPI(n):
    sum = 0.0
    for k in range(n):
        sum += 1 / ((4*k + 1) * (4*k + 3))
    return 8 * sum

def EulerPI(n):
    sum = 0.0
    for k in range(1,n):
        sum += 1 / (k ** 2)
    return (6 * sum) ** 0.5
```

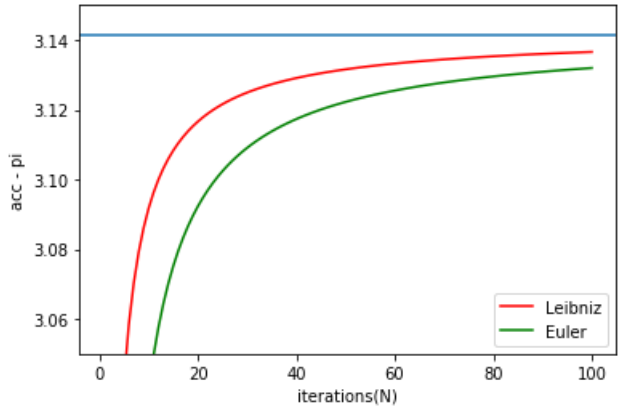
In [5]:

```
N = int(input("iterations(N): "))
print("Leibniz:",LeibnizPI(N))
print("Euler:  ",EulerPI(N))
```

```
iterations(N): 100
Leibniz: 3.1365926848388144
Euler:   3.1319807472443624
```

In [8]:

```
import pylab as plt
import math

LeibnizResult = []; EulerResult = []
t = range(1, N+1)

for k in t:
    LeibnizResult.append(LeibnizPI(k))
    EulerResult.append(EulerPI(k))

plt.plot(t, LeibnizResult, 'r')
plt.plot(t, EulerResult, 'g')
plt.axhline(y=math.pi)
plt.xlabel('iterations(N)'); plt.ylabel('acc - pi')
plt.ylim(3.05, 3.15)
plt.legend(['Leibniz','Euler'])
plt.show()
```



7. Loop

Problem 1

Make random integers - Dice function

In [1]:

```
import numpy as np

def Dice(N):
    num = []

    for _ in range(N):
        num.append(np.random.randint(1, 7))

    return num
```

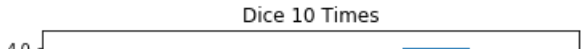
In [50]:

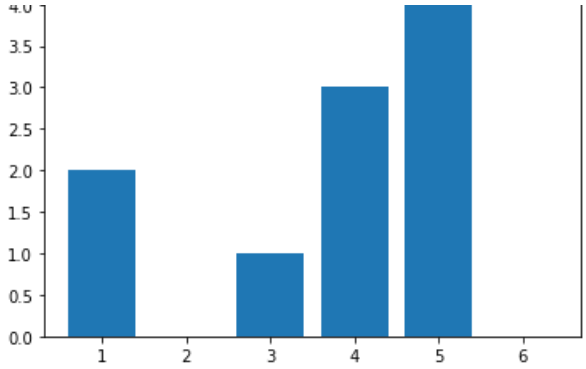
```
result = [] # Dice 결과 저장

for _ in range(10, 100+10, 10): # 10~100까지 10 증가해서 결과 저장
    result.append(Dice(_))

cnt = []

for i in range(0, 10, 1): # Dice 개수(10개+)마다 1~6 경우의 개수 각각 저장
    tmp = [0, 0, 0, 0, 0, 0]
    for j in range(0, len(result[i]), 1):
        for k in range(1, 7, 1):
            if result[i][j] == k:
                tmp[k-1] += 1
    cnt.append([(i+1)*10, tmp])

print(cnt)
print(cnt[0][1])

freq = [] # 주사위 던지는 횟수에 따른 1~6 빈도수를 저장할 배열

for i in range(0, 10, 1):
    tmp = []
    for j in cnt[i][1]:
        tmp.append(round(j/((i+1)*10),3))
    freq.append([(i+1)*10, tmp])

print(freq)
```

```
[[10, [2, 0, 1, 3, 4, 0]], [20, [5, 5, 2, 2, 5, 1]], [30, [5, 1, 11, 4, 4, 5]], [40, [8, 9, 7, 6,
8, 2]], [50, [13, 5, 8, 3, 8, 13]], [60, [5, 11, 13, 8, 14, 9]], [70, [11, 13, 12, 14, 11, 9]], [8
0, [13, 13, 17, 17, 10, 10]], [90, [15, 19, 16, 18, 15, 7]], [100, [17, 17, 12, 21, 19, 14]]]
[2, 0, 1, 3, 4, 0]
[[10, [0.2, 0.0, 0.1, 0.3, 0.4, 0.0]], [20, [0.25, 0.25, 0.1, 0.1, 0.25, 0.05]], [30, [0.167,
0.033, 0.367, 0.133, 0.133, 0.167]], [40, [0.2, 0.225, 0.175, 0.15, 0.2, 0.05]], [50, [0.26, 0.1,
0.16, 0.06, 0.16, 0.26]], [60, [0.083, 0.183, 0.217, 0.133, 0.233, 0.15]], [70, [0.157, 0.186, 0.1
71, 0.2, 0.157, 0.129]], [80, [0.163, 0.163, 0.212, 0.212, 0.125, 0.125]], [90, [0.167, 0.211, 0.1
78, 0.2, 0.167, 0.078]], [100, [0.17, 0.17, 0.12, 0.21, 0.19, 0.14]]]
```

In [80]:

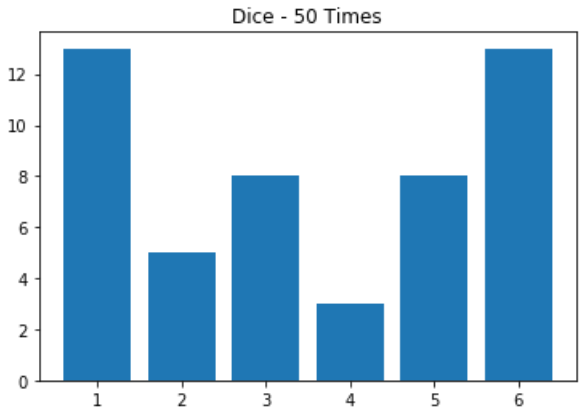
```
import pylab as plt

x = range(1, 7)
W = cnt[0][1]
plt.bar(x, W)
plt.title("Dice - 10 Times")
plt.show()
```





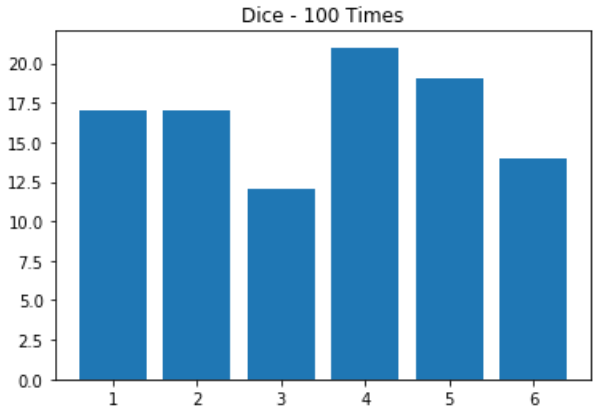
In [81]:

```
W = cnt[4][1]
plt.bar(x, W)
plt.title("Dice - 50 Times")
plt.show()
```



In [82]:

```
W = cnt[9][1]
plt.bar(x, W)
plt.title("Dice - 100 Times")
plt.show()
```



Problem 2

Black Jack

In [1]:

```
import numpy as np

def GetCard():
    '''generate rand num 0~10'''
    return np.random.randint(0,10)
```

In [2]:

```
def BlackJack():
    '''BlackJack - Make number closet to 21'''
    USER = 0
    BOT = 0
    KEEP = 0
    while KEEP == 0:
        USER += GetCard()
        BOTCARD = GetCard()

        if BOT + BOTCARD <= 21:
            BOT += BOTCARD

        if USER > 21:
            print(USER,'was over than 21.. You lose')
            break

        print('Your sum: ',USER)
        print('0: Draw, 1: Not')
        KEEP = int(input()) # User's decision: Keep getting card

    if USER > BOT and USER <= 21:
        if USER == 21:
            print("BLACKJACK! YOU WIN!")
        else:
            print("YOU WIN! - YOU:", USER, "BOT:", BOT)
    elif USER == BOT and USER <= 21:
        print("DRAW!")
    else:
        print("You lose.. - YOU:", USER, "BOT:", BOT)
```

In [3]:

```
BlackJack()
```

```
Your sum:  3
0: Draw, 1: Not
0
Your sum:  11
0: Draw, 1: Not
0
Your sum:  15
0: Draw, 1: Not
0
24 was over than 21.. You lose
You lose.. - YOU: 24 BOT: 16
```

In [ ]:

```

```

Problem 3

write a program that fits a straight line to data -> f(x) = ax + b

In [2]:

```
D = {(0, 0.5),(1, 2.0),(2,1.0),(3,1.5),(4,7.5)} # init data (x,y)
D = list(D) # change set to list
for _ in range(0,5): # also change inner
    D[_] = list(D[_])
print(D)
```

```
[[1, 2.0], [4, 7.5], [0, 0.5], [3, 1.5], [2, 1.0]]
```

make a function compute_error(a, b) that computes error between the straight line f(x) = ax+b and D

In [3]:

```
def compute_error(a, b):
    e = 0
    for _ in range(5):
        e += (a * D[_][0] + b - D[_][1]) ** 2 # e = SIGMA(i=1~5) ax + b - y
    return e
```

Search for a and b such that e is minimized.

In [4]:

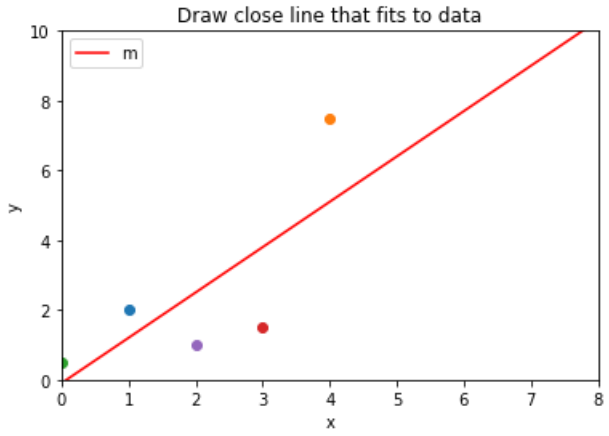
```
# a: 양수, b: 음수 (1, 3사분면 지남)

min = compute_error(1, -1) # error 초기 최소값
min_ab = [1, -1] # 최소값일 경우의 a, b

iter = 0.1 # 가중치 설정 (It's TRADEOFF)

for i in range(0, 1000, 1):

    if i % 100 == 0:
        print((i+100)/10,"%..")

    for j in range(-1000, 1):
        E = compute_error(i*iter, j*iter)
        if E < min:
            min = E; min_ab[0] = i*iter; min_ab[1] = j*iter

print("E min:", round(min, 3),"at [a, b]:",min_ab)
```

```
10.0 %..
20.0 %..
30.0 %..
40.0 %..
50.0 %..
60.0 %..
70.0 %..
80.0 %..
90.0 %..
100.0 %..
E min: 14.3 at [a, b]: [1.3, -0.1]
```

Plot a straight line f(x) = ax + b

In [55]:

```
import pylab as plt
import numpy as np
# a: min_ab[0], b: min_ab[1]
x = np.arange(0, 11, 1)
y = []
for _ in range(0, 11, 1):
    y.append(min_ab[0] * x[_] + min_ab[1])
plt.ylim(0, 10)
plt.xlim(0, 8)
plt.plot(x, y, 'r')
plt.title('Draw close line that fits to data')
plt.xlabel('x'); plt.ylabel('y')
plt.legend('m')

#plt.scatter - 그래프에 점찍기
for _ in range(5):
    plt.scatter(D[_][0], D[_][1])
```



8. Sets and Dicts

Problem 1

In [2]:

```
import random

def countNum():
    nums = dict()
    for i in range(100):
        num = random.randint(1,20)
        if num in nums:
            nums[num] = nums[num] + 1
        else:
            nums[num] = 1

    for i in nums.items():
        print(i)
countNum()
```

```
(1, 8)
(12, 5)
(16, 5)
(4, 9)
(5, 5)
(11, 5)
(9, 3)
(15, 4)
(8, 4)
(20, 6)
(10, 7)
(3, 2)
(2, 5)
(14, 5)
(7, 7)
(6, 2)
(17, 6)
(13, 6)
(18, 3)
(19, 3)
```

Problem 2

In [4]:

```
def count_values_1(dic):
    vs = dic.values()
    vs = list(vs)
    count = dict()

    for i in vs:
        if i in count:
            count[i] = count[i] + 1
        else:
            count[i] = 1

    return len(count)

def count_values_2(dic):
    vs = dic.values()
    vs = set(vs)

    return len(vs)

temp = {'red' : 1, 'green' : 1, 'blue' : 2}
print(count_values_1(temp))
print(count_values_2(temp))
```

```
2
2
```

Problem 3

In [5]:

```
def normal_to_sparse(vec):
    sps = dict()

    for i in range(0, len(vec)):
        if vec[i] == 0 : continue
        else:
            sps[i] = vec[i]

    return sps

def change_sign(dic):
    keys = dic.keys()
    for i in keys:
        dic[i] = -dic[i]
    return dic

def add_vector(dic1, dic2):
    rdic = dict()
    for i in dic1.keys():
        for j in dic2.keys():
            if(i == j):
                rdic[i] = dic1[i] + dic2[i]
    for i in dic1.keys():
        if i not in rdic:
            rdic[i] = dic1[i]
    for i in dic2.keys():
        if i not in rdic:
            rdic[i] = dic2[i]
    return rdic

def minus_vector(dic1, dic2):
    return add_vector(dic1, change_sign(dic2))



vec = [1,0,0,0,0,0,3,0,0,0]
print(normal_to_sparse(vec))
print(change_sign(normal_to_sparse(vec)))
print(add_vector({0:1, 6:3}, {1:2, 6:3}))
print(minus_vector({0:1, 6:3}, {1:2, 6:3}))
```

```
{0: 1, 6: 3}
{0: -1, 6: -3}
{6: 6, 0: 1, 1: 2}
{6: 0, 0: 1, 1: -2}
```