Open Source SW Utilization

Department of Software
32153180
Lee Sang Min

1. Basics in Python

Problem 1

What are the values of the following expressions, assuming that *n* = 17 and *m* = 18 ?

1. `n // 10 + n % 10`
2. `n % 2 + m % 2`
3. `(m + n) // 2`
4. `(m + n) / 2.0`
5. `int(0.5 * (m + n))`
6. `int(round(0.5 * (m + n))`

In [1]:

```
n = 17
m = 18

print('n//10 + n%10 :', n//10 + n%10)
print('n%2 + m%2 :', n%2 + m%2)
print('(m+n)//2 :', (m+n) // 2)
print('(m+n)//2.0 :', (m+n) // 2.0)
print('int(0.5*(m+n)) :', int(0.5 * (m+n)))
print('int(round(0.5*(m+n))) :', int(round(0.5 * (m+n))))
```

```
n//10 + n%10 : 8
n%2 + m%2 : 1
(m+n)//2 : 17
(m+n)//2.0 : 17.0
int(0.5*(m+n)) : 17
int(round(0.5*(m+n))) : 18
```

Problem 2

What are the values of the following expressions, assuming that *s* ="Hello" and *t* = "World" ?

1. `len(s) + len(t)`
2. `s[1] + s[2]`
3. `s[len(s) // 2]`
4. `s + t`
5. `t + s`
6. `s * 2`

In [2]:

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

print('len(s)+len(t) :', len(s) + len(t))
print('s[1]+s[2] :', s[1] + s[2])
print('s[len(s)//2] :', s[len(s) // 2])
print('s+t :', s + t)
print('t+s :', t + s)
print('s*2 :', s * 2)
```

```
len(s)+len(t) : 11
s[1]+s[2] : He
s[len(s)//2] : l
s+t :  HelloWorld
t+s : World Hello
s*2 :  Hello Hello
```

Problem 3

What is a projectile ?

- A projectile is an object that is given some energy to cause it to go up, reach a maximum height, and then fall down to the ground.
- The projectile can be a soccer ball, rock, bullet, baseball, or cannon ball.
- These objects all take on the same characteristic arc as they fly through the air.
- Soccer balls get kicked, rocks and baseballs get tossed, and bullets and cannon balls all fly through the air with a characteristic upside-down u-shape. Design a program to compute the height of a ball in vertical motion.

$$h(t) = v_0 \times -\frac{1}{2} \times g \times t^2$$

  - *g* : Earth's gravity is different than the gravity on other planets. That force is 32 feet/sec every second.
  - v_0 : The starting upward velocity will differ for each problem.
  - *d* : The initial height of the projectile is important. If the projectile starts up high, it can get even higher after it is launched.

In [3]:

```
v0 = 100        # initial velocity
g = 32          # acceleration of gravity (32 feet/sec)
t = 3.2         # time
d = 50          # initial vertical position

# compute the vertical position
h = -0.5 * g * t**2 + v0 * t + d
print(round(h, 2), 'feet')
```

```
206.16 feet
```

2. Data types and Strings

Problem 1

Practice how variables refer to Python objects. Test each code line to understand the results of Python objects.

```
1. a = -11
2. b = 11
3. c = 9.0
4. d = b / a
5. e = c / a
6. s = 'b / a = %g' % (b / a)
```

In [4]:

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

print('a : {}'.format(type(a)))
print('b : {}'.format(type(b)))
print('c : {}'.format(type(c)))
print('d : {}'.format(type(d)))
print('e : {}'.format(type(e)))
print("s : {}".format(type(s)))
```

```
a : <class 'int'>
b : <class 'int'>
c : <class 'float'>
d : <class 'float'>
e : <class 'float'>
s : <class 'str'>
```

Problem 2

Convert between object types.

```
1. a = 3
2. b = float(a)
3. c = 3.9
4. d = int(c)
5. e = round(c)
6. f = int(round(c))
7. d = str(c)
8. e = '-4.2'
9. f = float(e)
```

In [5]:

```
a = 3
b = float(a)
c = 3.9
d = int(c)
e = round(c)
f = int(round(c))

print('a : {}'.format(type(a)))
print('b : {}'.format(type(b)))
print('c : {}'.format(type(c)))
print('d : {}'.format(type(d)))
print('e : {}'.format(type(e)))
print('f : {}'.format(type(f)))

d = str(c)
e = '-4.1'
f = float(e)

print('d : {}'.format(type(d)))
print('e : {}'.format(type(e)))
print('f : {}'.format(type(f)))
```

```
a : <class 'int'>
b : <class 'float'>
c : <class 'float'>
d : <class 'int'>
e : <class 'int'>
f : <class 'int'>
d : <class 'str'>
e : <class 'str'>
f : <class 'float'>
```

Problem 4

Compute the equation.

$$sinh(x) = \frac{1}{2}(e^x - e^{-x})$$

- import **math** and use **sinh, pi, e, exp**

In [6]:

```
import math

x = int(input('input x : '))
result1 = round(math.sinh(x), 2)
result2 = round(0.5 * (math.exp(x) - math.exp(-x)), 2)

print('sinh({}) : {}'.format(x, result1))
print('1/2(e^{} - e^-{}) : {}'.format(x, x, result2))
```

```
input x : 5
sinh(5) : 74.2
1/2(e^5 - e^-5) : 74.2
```

Problem 5

Throw a ball with velocity v_0, at an angle θ with the horizontal, from the point (*x* = 0, *y* = y_0). The trajectory of the all is a parabola. We neglect air resistance.

$$y = xtan\theta - \frac{1}{2v_0^2}\frac{gx^2}{cos^2\theta} + y_0$$

- import **math** and use **cos**
- Initialize input data (v_0, θ, y_0), where v_0's unit is km/h and θ is degree
- *g* takes 9.81m/s^2
- Compute and print *y*

In [7]:

```
import math

v0 = float(input('input velocity : '))
seta = int(input('input angle : '))
y0 = int(input('input y0 : '))
x = int(input('input x : '))

g = 9.81

y = x*math.tan(seta) - (g*pow(x, 2)) / (2*pow(v0, 2)*pow(math.cos(seta), 2)) + y0

print('y : %.2f' % y)
```

```
input velocity : 13.24
input angle : 150
input y0 : 7
input x : 5
y : 0.46
```

Problem 6

Let *p* be a bank's interest rate in percent per year. After *n* years, an initial amount *A* has taken grown to

$$A(1 + \frac{p}{100})^n$$

- Make a program for computing how much money 10,000,000 have grown to after three years with 5% interest rate.

In [8]:

```
p = float(input('input interest rate : '))
n = int(input('input year : '))
A = int(input('input initial amount : '))

result = A*(1+(p/100))**n

print('total (initial amount + interest) :', round(result))
print('interest (total - initial amount) :', round(result-A))
```

```
input interest rate : 5
input year : 3
input initial amount : 10000000
total (initial amount + interest) : 11576250
interest (total - initial amount) : 1576250
```

3. Module

Problem 1

There are various geometries. Design modules to calculate area, perimeter, volume, or surface related to a chosen geometry.

- Divide the related operations into two groups or more
- Design and test the designed module
- Design the related functions by function
- Write some informative comment to every function
- Design your drive module for user's input using if-then statements

In [9]:

```
# circle module
import math

class circle:
    def __init__(self, r):
        self.r = r
    def perimeter(self):
        return round(2 * math.pi * self.r, 2)
    def area(self):
        return round(math.pi * self.r**2, 2)

class circular_sector:
    def __init__(self, r, seta):
        self.r = r
        self.seta = seta
    def length(self):
        return round(math.pi * self.r * self.seta/180, 2)
    def area(self):
        return round(math.pi * self.r**2 * self.seta/360, 2)

class circular_ring:
    def __init__(self, r, R):
        self.r = r
        self.R = R
    def area(self):
        return round(math.pi * (self.R**2 - self.r**2), 2)
```

In [10]:

```
# triangle module
import math

class triangle:
    def __init__(self, a, b, c, h):
        self.a = a
        self.b = b
        self.c = c
        self.h = h
    def perimeter(self):
        return round(self.a + self.b + self.c, 2)
    def area(self):
        return round(0.5 * self.b * self.h, 2)

class pythagorean:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c
    def hypotenuse(self):
        return round(math.sqrt(self.a**2 + self.b**2), 2)
```

In [11]:

```
# rectangle module
class square:
    def __init__(self, s):
        self.s = s
    def perimeter(self):
        return round(4 * self.s, 2)
    def area(self):
        return round(self.s**2, 2)

class rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def perimeter(self):
        return round(2*self.a + 2*self.b, 2)
    def area(self):
        return round(self.a * self.b, 2)

class parallelogram:
    def __init__(self, a, b, h):
        self.a = a
        self.b = b
        self.h = h
    def perimeter(self):
        return round(2*self.a + 2*self.b, 2)
    def area(self):
        return round(self.b * self.h, 2)

class trapezoid:
    def __init__(self, a, b, c, d, h):
        self.a = a
        self.b = b
        self.c = c
        self.d = d
        self.h = h
    def perimeter(self):
        return round(self.a + self.b + self.c + self.d, 2)
    def area(self):
        return round(self.h * (self.a + self.b)/2, 2)
```

In [12]:

```
# dimensional module
import math

class sphere:
    def __init__(self, r):
        self.r = r
    def surface(self):
        return round(4 * math.pi * self.r**2, 2)
    def volume(self):
        return round(4 * math.pi * self.r**3 / 3, 2)

class rectangular_box:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c
    def area(self):
        return round(2*self.a*self.b + 2*self.a*self.c + 2*self.b*self.c, 2)
    def volume(self):
        return round(self.a * self.b * self.c, 2)

class circular_cone:
    def __init__(self, r, h, s):
        self.r = r
        self.h = h
        self.s = s
    def area(self):
        return round(math.pi * self.r**2 + math.pi * self.r * self.s, 2)
    def surface(self):
        return round(math.sqrt(self.r**2 + self.h**2), 2)
    def volume(self):
        return round(1/3 * math.pi * self.r**2 * self.h, 2)

class cube:
    def __init__(self, l):
        self.l = l
    def area(self):
        return round(6 * self.l**2, 2)
    def volume(self):
        return round(self.l**3, 2)

class cylinder:
    def __init__(self, r, h):
        self.r = r
        self.h = h
    def area(self):
        return round(2 * math.pi * self.r * (self.r + self.h), 2)
    def volume(self):
        return round(math.pi * self.r**2 * self.h, 2)

class frustum_cone:
    def __init__(self, r, h, R):
        self.r = r
        self.h = h
        self.R = R
    def volume(self):
        return round(1/3 * math.pi * self.h * (self.r**2 + self.r*self.R + self.R**2), 2)
```

In [13]:

```
import circle
import triangle
import rectangle
import dimensional

r, R, seta = map(int, input('input r, R, seta : ').split())
a, b, c, d = map(int, input('input a, b, c, d : ').split())
s, h = map(int, input('input s, h : ').split())

c1, c2, c3 = circle.circle(r), circle.circular_sector(r, seta), circle.circular_ring(r, R)
t1, t2 = triangle.triangle(a, b, c, h), triangle.pythagorean(a, b, c)
r1, r2, r3, r4 = rectangle.square(s), rectangle.rectangle(a, b), ₩
                    rectangle.parallelogram(a, b, h), rectangle.trapezoid(a, b, c, d, h)
d1, d2, d3, d4, d5, d6 = dimensional.sphere(r), dimensional.rectangular_box(a, b, c), ₩
                         dimensional.cube(s), dimensional.cylinder(r, h), ₩
                         dimensional.circular_cone(r, h, s), dimensional.frustum_cone(r, h, R
)

print('₩n<< CIRCLE >>')
print('perimeter :',c1.perimeter(), '/ area :', c1.area())
print('₩n<< CIRCLE SECTOR >>')
print('length :', c2.length(), '/ area :', c2.area())
print('₩n<< CIRCLE RING >>')
print('area :', c3.area())
print('₩n<< TRIANGLE >>')
print('perimeter :', t1.perimeter(), ' / area :', t1.area())
print('₩n<< PYTHAGOREAN THEOREM >>')
print('c :', t2.hypotenuse())
print('₩n<< SQUARE >>')
print('perimeter :', r1.perimeter(), ' / area :', r1.area())
print('₩n<< TECTANGLE >>')
print('perimeter :', r2.perimeter(), ' / area :', r2.area())
print('₩n<< PARALLELOGRAM >>')
print('perimeter :', r3.perimeter(), ' / area :', r3.area())
print('₩n<< TRAPEZOID >>')
print('perimeter :', r4.perimeter(), ' / area :', r4.area())
print('₩n<< SPHERE >>')
print('surface :', d1.surface(), ' / volume :', d1.volume())
print('₩n<< RECTANGULAR BOX >>')
print('area :', d2.area(), ' / volume :', d2.volume())
print('₩n<< CUBE >>')
print('area :', d3.area(), ' / volume :', d3.volume())
print('₩n<< CYLINDER >>')
print('area :', d4.area(), ' / volume :', d4.volume())
print('₩n<< RIGHT CIRCULAR CONE >>')
print('area :', d5.area(), ' / surface :', d5.surface(), ' / volume :', d5.volume())
print('₩n<< FRUSTUM OF A CONE >>')
print('volume :', d6.volume())
```

```
input r, R, seta : 3 6 120
input a, b, c, d : 3 4 2 7
input s, h : 10 12

<< CIRCLE >>
perimeter : 18.85 / area : 28.27

<< CIRCLE SECTOR >>
length : 6.28 / area : 9.42

<< CIRCLE RING >>
area : 84.82

<< TRIANGLE >>
perimeter : 9  / area : 24.0

<< PYTHAGOREAN THEOREM >>
c : 5.0

<< SQUARE >>
perimeter : 40  / area : 100

<< TECTANGLE >>
perimeter : 14  / area : 12

<< PARALLELOGRAM >>
perimeter : 14  / area : 48

<< TRAPEZOID >>
perimeter : 16  / area : 42.0

<< SPHERE >>
surface : 113.1  / volume : 113.1

<< RECTANGULAR BOX >>
area : 52  / volume : 24

<< CUBE >>
area : 600  / volume : 1000

<< CYLINDER >>
area : 282.74  / volume : 339.29

<< RIGHT CIRCULAR CONE >>
area : 122.52  / surface : 12.37  / volume : 113.1

<< FRUSTUM OF A CONE >>
volume : 791.68
```

In [14]:

```
# euclidean module
from scipy.spatial import distance

def dist(u, v):
    return round(distance.euclidean(u, v), 2)
```

In [15]:

```
# manhattan module
from scipy.spatial import distance

def dist(u, v):
    return round(distance.cityblock(u, v), 2)
```

In [16]:

```
# hamming module
from scipy.spatial import distance

def dist(u, v):
    return round(distance.hamming(u, v), 2)
```

In [17]:

```
# cosine module
from scipy.spatial import distance

def dist(u, v):
    return round(distance.cosine(u, v), 2)
```

In [18]:

```
# chebyshev module
from scipy.spatial import distance

def dist(u, v):
    return round(distance.chebyshev(u, v), 2)
```

In [19]:

```
import numpy as np
import euclidean
import manhattan
import hamming
import cosine
import chebyshev

N = int(input('input N-dimension : '))
u = [np.random.randint(10) for x in range(N)]
v = [np.random.randint(10) for x in range(N)]
print('u vector :', u)
print('v vector :', v)

print('Euclidean distance :', euclidean.dist(u, v))
print('Manhattan distance :', manhattan.dist(u, v))
print('Hamming distance :', hamming.dist(u, v))
print('Cosine distance :', cosine.dist(u, v))
print('Chebyshev distance :', chebyshev.dist(u, v))
```

```
input N-dimension : 3
u vector : [4, 0, 0]
v vector : [9, 7, 7]
Euclidean distance : 11.09
Manhattan distance : 19
Hamming distance : 1.0
Cosine distance : 0.33
Chebyshev distance : 7
```

4. Lists

Problem 1

One of the most important mathematical problems has been to find the area of a polygon. We have a polygon as depicted below. The vertices of polygon *P* have coordinates (x1, y1), (x2, y2), · · · , (xn, yn), numbered either in a clockwise or counter clockwise way. The area *P* of the polygon can be computed by just knowing the boundary coordinates:

$$P = \frac{1}{2}|(x_1y_2 + x_2y_3 + \ldots + x_{n-1}y_n + x_ny_1) - (y_1x_2 + y_2x_3 + \ldots + y_{n-1}x_n + y_nx_1)|$$

Assume that *x* and *y* are either lists or arrays. Implement the function to compute the area of a polygon and test your function on a triangular, a quadrilateral, and a pentagon where you can calculate the area by alternative methods for computation.

In [20]:

```
n = int(input('input the number of vertex : '))
arr = []
for i in range(n):
    x, y = map(int, input('input x, y : ').split())
    arr.append([x, y])

area = 0
for i in range(n):
    if i == n-1:
        area += arr[i][0]*arr[0][1] - arr[i][1]*arr[0][0]
    else:
        area += arr[i][0]*arr[i+1][1] - arr[i][1]*arr[i+1][0]

print('area of polygon', abs(area)*0.5)
```

```
input the number of vertex : 5
input x, y : 1 1
input x, y : 2 4
input x, y : 4 5
input x, y : 5 2
input x, y : 3 1
area of polygon 10.0
```

Problem 2

Consider two functions $f(x) = x$ abd $g(x) = x^2$ on the interval [-4, 4]. Write a program that finds approximately for which values of *x* the two graphs cross, i.e., $f(x) = g(x)$. Do this by Considering N equally distributed points on the interval, at each point checking whether $|f(x) - g(x)| < \epsilon$ with a fixed value ϵ. Let N and ϵ be user input to the program and let the result be printed to screen.

- Run your program with N = 400 and ϵ = 0.01.
- Explain the output from your program and try other values of N with fixed ϵ.

In [21]:

```
N = int(input('input N : '))
e = float(input('input e : '))

arr = []
result = []
sum = -4

for i in range(N+1):
    arr.append(round(sum, 2))
    sum += (8/N)

    if abs(sum - sum**2) < e:
        result.append(round(sum, 2))

print('x :', result)
```

```
input N : 400
input e : 0.01
x : [0.0, 1.0]
```

Problem 3

Up through history, great minds have developed different computational schemes for the number π. There are two schemes: one by Leibniz (1646-1716) and one by Euler (1707 - 1783). The scheme by Leibniz may be written

$$\pi = 8 \sum_{k=0}^{\infty} \frac{1}{(4k+1)(4k+3)}$$

with the Euler scheme appears as

$$\pi = \sqrt{6 \sum_{k=1}^{\infty} \frac{1}{k^2}}$$

If only the first N terms of each sum are used as an approximation to π, each modified scheme will have computed π with some error. Your program should also print out the final error achieved with both schemes, i.e. when the number of terms is N. Run the program with N = 100 and explain briefly what the graphs show.

- Write a program that takes N as input from the user.
- Plot the values and errors of both schemes as the number of iterations approaches N when N is 100.
- Explain what your graph shows.

In [22]:

```
import math

N = int(input('input N : '))
leibniz = euler = 0

for i in range(N):
    leibniz += (8 / (4*i+1) / (4*i+3))

for i in range(1, N):
    euler += (6 / i**2)
euler = math.sqrt(euler)

print('math pi :', round(math.pi, 4))
print('Leibniz pi :', round(leibniz, 4))
print('Euler pi :', round(euler, 4))
```

```
input N : 100
math pi : 3.1416
Leibniz pi : 3.1366
Euler pi : 3.132
```

5. Loop statements

Problem 1

Write a program that takes a positive integer N as input and draws N random integers in the interval [1, 6] (both ends inclusive). N increase by 10 to 100. Answer the following questions.

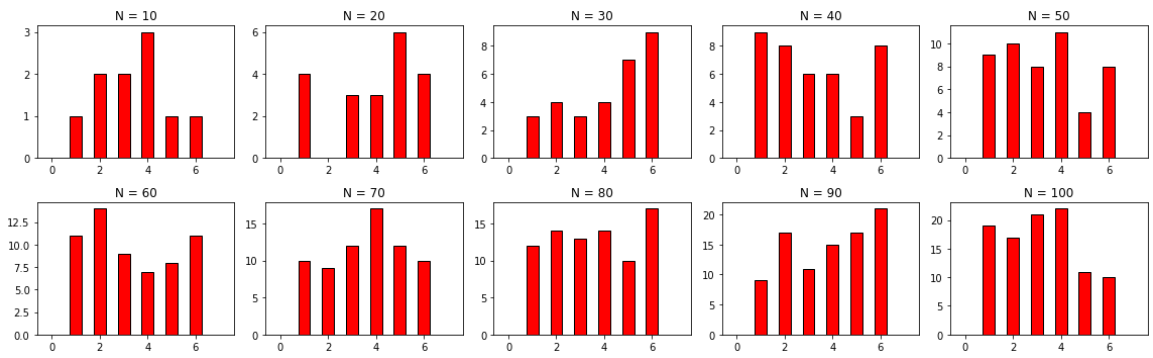
- Count the frequency of each number and compute the fractions.
- Plot both the frequency and the fraction of each number.
- Discuss the changing trend as N increases.

Use **random.randint(1, 6)** from module **random** or **numpy.random.randint(1, 7)** from module **numpy**.

In [25]:

```
import numpy as np
from matplotlib import pyplot as plt

N = int(input('input N : '))

fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(16, 5))

row = 0
col = 0
for step in range(10, N+1, 10):
    count = [0 for x in range(8)]
    base = [x for x in range(8)]
    rand = [np.random.randint(1, 7) for _ in range(step)]

    for i in range(step):
        for j in range(step):
            if rand[j] == i:
                count[i] += 1

    if col == 5:
        row = 1
        col = 0
    ax[row][col].bar(base, count, width=0.5, color='r', edgecolor='black')
    ax[row][col].set(title = 'N = %d' % step)
    col += 1

plt.tight_layout()
plt.show()
```

```
input N : 100
```



Problem 2

Consider some game where each participant draws a series of random integers evenly distributed from 0 to 10, with the aim of getting the sum as close as possible to 21, but not larger than 21. You are out of the game if the sum passes 21. After each draw, you are told the number and is asked whether you want another draw or not. The one coming closest to 21 is the winner. Implement this game.
Use **random.randint(0, 10)** from module **random** or **numpy.random.randint(0, 11)** from module **numpy**.

In [27]:

```
import numpy as np

sum = 0
while True:
    rand = np.random.randint(0, 11)
    sum += rand

    if sum > 21:
        print('₩nOops.. computer wins the game, your total number is', sum)
        break

    print('your number is', rand, 'and total is', sum)
    answer = input('if you draw more card ? (yes/no) ')

    if answer == 'yes':
        continue
    else:
        print('₩nCongratulation ! your total number is', sum)
        break
```

```
your number is 8 and total is 8
if you draw more card ? (yes/no) yes
your number is 7 and total is 15
if you draw more card ? (yes/no) yes

Oops.. computer wins the game, your total number is 25
```

Problem 3

There are some data *D* which are collected from a device.

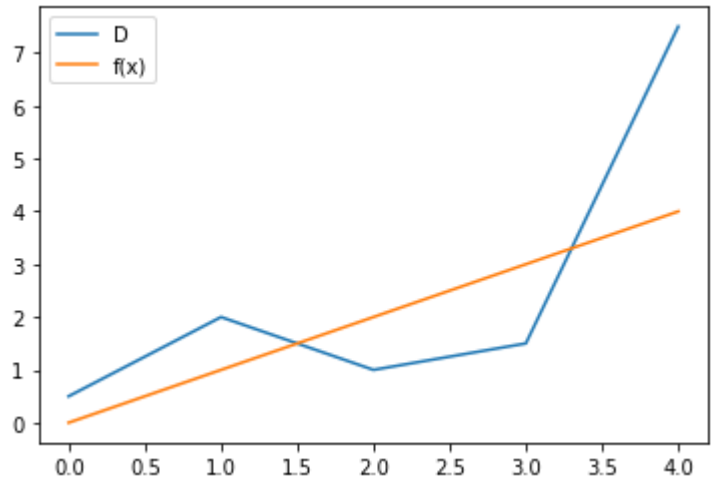
D = {(0, 0.05), (1, 2.0), (2, 1.0), (3, 1.5), (4, 7.5)}

- Given a and b, make a function compute*error(a, b, y) that computes the error between the straight line* $f(x) = ax + b$ *and D. $$e = \sum{i=1}^{5}(ax{i} + b - y{i})$$*
- Plot *a* straight line f(x) given *a* and *b*.
- Search for *a* and *b* such that *e* is minimized.

In [28]:

```
from matplotlib import pyplot as plt

D = [(0, 0.5), (1, 2.0), (2, 1.0), (3, 1.5), (4, 7.5)]
a, b = map(float, input('input a, b : ').split())

x = []
y = []
def compute_error(a, b, y):
    fx = []
    e = 0

    for i in range(len(D)):
        x.append(D[i][0])
        y.append(D[i][1])
        fx.append(a*x[i] + b)
        e += (a*x[i] + b - y[i])**2

    print('error e :', round(e, 2))

    plt.plot(x, y)
    plt.plot(x, fx)
    plt.legend(['D', 'f(x)'])
    plt.show()

compute_error(a, b, y)
```

```
input a, b : 1 0
error e : 16.75
```



6. Sets and Dictionaries

Problem 1

Design and implement a function that takes a set of integers as its input argument and returns a set of those integers that occur two or more times in the list. The input integers are generated randomly in [1, 20].

In [29]:

```
import numpy as np

print('making random 20 number from 1 to 20 ...')
arr = [np.random.randint(1, 21) for _ in range(20)]
print('random 20 number :', arr)
temp = [0 for _ in range(20)]

def duplicate(arr):
    for i in range(20):
        for j in range(20):
            if arr[j] - 1 == i:
                temp[i] += 1

    result = []
    for i in range(20):
        if temp[i] >= 2:
            result.append(i + 1)

    return result

print('₩nintegers that occur two or more times :', duplicate(arr))
```

```
making random 20 number from 1 to 20 ...
random 20 number : [2, 16, 4, 11, 5, 4, 14, 15, 10, 1, 8, 5, 14, 2, 11, 7, 5, 19,
4, 8]

integers that occur two or more times : [2, 4, 5, 8, 11, 14]
```

Problem 2

The keys in a dictionary are guaranteed to be unique, but the values are not. Write a function that takes a single dictionary as an argument and returns the number of distinct values it contains. For example, given the input {'red': 1, 'green': 1, 'blue': 2}, your function should return 2.

- Design and implement a function called **count_values1** that takes a single dictionary as an argument and returns the number of distinct values it contains.
- Based on set object, design and implement a function called **count_values2** that can do the same work.

In [30]:

```
problem = {'red': 1, 'green': 1, 'blue': 2, 'pink': 1, 'purple': 2, 'grey': 3}
print(problem)

def count_values1(problem):
    val = list(problem.values())
    temp = []

    for i in range(len(val)):
        if val[i] not in temp:
            temp.append(val[i])
    return temp

def count_values2(problem):
    val = set(problem.values())
    return val

print('distinct values :', count_values1(problem))
print('distinct values :', count_values2(problem))
```

```
{'red': 1, 'green': 1, 'blue': 2, 'pink': 1, 'purple': 2, 'grey': 3}
distinct values : [1, 2, 3]
distinct values : {1, 2, 3}
```

Problem 3

A sparse vector is a vector whose entries are almost all zero, like [1, 0, 0, 0, 0, 0, 3, 0, 0, 0]. Storing all those zeros in a list wastes memory, so programmers often use dictionaries instead to keep track of just the nonzero entries. For example, the vector shown earlier would be represented as {0:1, 6:3}, because the vector it is meant to represent has the value 1 at index 0 and the value 3 at index 6.

1. Design and implement a function called **normal_to_sparse** that converts a normal vector to its sparse vector.
2. Design and implement a function called **change_sign** that takes a sparse vector and returns its negative vector.
3. Design and implement a function called **add_vector** that takes two sparse vectors, adds them and returns a sparse vector representing their sum.
4. Extend the function **minus_vector** that takes two sparse vectors, subtracts them and return the result.

In [31]:

```
first = [1, 0, 0, 0, 0, 0, 3, 0, 0, 0]
second = {1: 2, 6: 2}
print('normal :', first)

def normal_to_sparse(list):
    temp = {}
    for idx, value in enumerate(list):
        if value != 0:
            temp[idx] = value
    return temp

def change_sign(dict):
    temp = {}
    for key, value in dict.items():
        temp[key] = dict[key] * -1
    return temp

def add_vector(one, two):
    one_list = [0 for x in range(10)]
    two_list = [0 for x in range(10)]
    for key, value in one.items():
        if key in range(10):
            one_list[key] = value
    for key, value in two.items():
        if key in range(10):
            two_list[key] = value

    total = [sum([one_list[i], two_list[i]]) for i in range(len(one_list))]
    return normal_to_sparse(total)

def minus_vector(one, two):
    return add_vector(one, change_sign(two))


sparse = normal_to_sparse(first)
minus_sparse = change_sign(sparse)
total_sparse = add_vector(sparse, second)
subtract_sparse = minus_vector(sparse, second)

print('₩nnormal to sparse :', sparse)
print('change sign', minus_sparse)
print('₩nshow two sparse vector :', sparse, second)
print('add :', total_sparse)
print('subtract :', subtract_sparse)
```

```
normal : [1, 0, 0, 0, 0, 0, 3, 0, 0, 0]

normal to sparse : {0: 1, 6: 3}
change sign {0: -1, 6: -3}

show two sparse vector : {0: 1, 6: 3} {1: 2, 6: 2}
add : {0: 1, 1: 2, 6: 5}
subtract : {0: 1, 1: -2, 6: 1}
```

7. Algorithms

Problem 1

A DNA sequence is a string made up of the letters A, T, G, and C. To find the complement of a DNA sequence, As are replaced by Ts, Ts by As, Gs by Cs, and Cs by Gs. For example, the complement of AATTGCCGT is TTAACGGCA.

1. Write a pseudo code in English of the algorithm that takes a DNA sequence and return its complement.
2. Test your algorithm.
3. Write a function named **complement** that takes a DNA sequence and returns the complement of it.

pseudo code

Input:
DNA sequence

Algorithm:
Function Complement(string)
for str in length(string)
if str is 'T' then str is 'A'
else if str is 'A' then str is 'T'
else if str is 'G' then str is 'C'
else if str is 'C' then str is 'G'
else str is ' ' (this case is empty space)
end

Output:
complement of a DNA sequence

In [32]:

```
def complement(str):
    temp = []
    for i in range(len(str)):
        if str[i] == 'T':
            temp.append('A')
        elif str[i] == 'A':
            temp.append('T')
        elif str[i] == 'G':
            temp.append('C')
        elif str[i] == 'C':
            temp.append('G')
        else:
            temp.append(' ')

    return "".join(temp)

code = input('input DNA sequence : ')
print('complement is', complement(code))
```

```
input DNA sequence : TTCCCATCAA GCCCTAGGGC TCCTCGTGGC TGCTGGGAGT TGTAGTCTGA ACGCTT
CTAT
complement is AAGGGTAGTT CGGGATCCCG AGGAGCACCG ACGACCCTCA ACATCAGACT TGCGAAGATA
```

Problem 2

Develop a function that finds the minimum or maximum value in a list, depending on the caller's request.

1. Write a loop (including initialization) to find both the minimum value in a list and that value’s index in one pass through the list.
2. Write a function named **min_index** that takes a list and returns a tuple containing the minimum value in the list and that value’s index in the list.
3. Write a function named **max_index** that takes a list and returns a tuple containing the maximum value in the list and that value’s index in the list.

In [33]:

```
import numpy as np

arr = [np.random.randint(100) for x in range(50)]

min_value = arr[0]
idx = 0
for i in range(len(arr)):
    if arr[i] <= min_value:
        min_value = arr[i]
        idx = i

def min_index(arr):
    return (min(arr), arr.index(min(arr)))

def max_index(arr):
    return (max(arr), arr.index(max(arr)))

print('(min value, min index) using loop :', (min_value, idx))
print('(min value, min index) using function :', min_index(arr))
print('(max value, max index) using function :', max_index(arr))
```

```
(min value, min index) using loop : (1, 11)
(min value, min index) using function : (1, 11)
(max value, max index) using function : (99, 39)
```