In [10]:

```
#Problem 2-Branching
from math import *
import numpy as np

def my_mult_operation(x,y,operation):
    if operation == "plus":
        output = np.add(x,y)
    elif operation == "minus":
        output = (x - y)
    elif operation == "mult":
        output = np.multiply(x,y)
    elif operation == "div":
        output = np.divide(x,y)
    elif operation == "pow":
        output = np.power(x,y)
    return output

x = np.array([1,2,3,4])
y = np.array([2,3,4,5])
print(my_mult_operation(x,y,"plus"))
print(my_mult_operation(x,y,"minus"))
print(my_mult_operation(x,y,"mult"))
print(my_mult_operation(x,y,"div"))
print(my_mult_operation(x,y,"pow"))
```

```
[3 5 7 9]
[-1 -1 -1 -1]
[ 2  6 12 20]
[0.5        0.66666667 0.75       0.8       ]
[   1    8   81 1024]
```

In [2]:

```
#Problem 3-Branching
from math import *
import numpy as np

def my_inside_triangle(x,y):
    x1 = 0
    y1 = 0
    x2 = 1
    y2 = 0
    x3 = 0
    y3 = 1

    def area(x1, y1, x2, y2, x3, y3):
        return abs((x1 * (y2 - y3) + x2 * (y3 - y1)
                + x3 * (y1 - y2)) / 2.0)

    A = area (x1, y1, x2, y2, x3, y3)
    A1 = area (x, y, x2, y2, x3, y3)
    A2 = area (x1, y1, x, y, x3, y3)
    A3 = area (x1, y1, x2, y2, x, y)

    if A1 == 0 or A2 ==0 or A3 == 0:
        return 'border'
    elif(A == A1 + A2 + A3):
        return 'inside'
    elif(A != A1 + A2 + A3):
        return 'outside'


my_inside_triangle(.5,.5)
```

Out[2]:

```
'border'
```

In [9]:

```
#Problem 7-Branching

def my_nuke_alarm(s1,s2,s3):

   if ((abs(s1-s2) > 10) or (abs(s2-s3) > 10) or (abs(s1-s3) > 10)):
    response =  "alarm!"
   else:
    response =  "normal"

   return response

print(my_nuke_alarm(94,96,90))
print(my_nuke_alarm(94,96,80))
print(my_nuke_alarm(100,96,90))
```

```
normal
alarm!
normal
```

In [11]:

```
#Problem 8-Branching
from math import *
import numpy as np
import cmath

def my_n_roots(a,b,c):
     if(b**2 > 4*a*c):
        n_roots = 2
        r1 = ((-b + np.sqrt(b**2 - 4*a*c))/(2*a))
        r2 = ((-b - np.sqrt(b**2 - 4*a*c))/(2*a))
        r = [r1, r2]
     elif(b**2 < 4*a*c):
        n_roots = -2
        r1 = ((-b + cmath.sqrt(b**2 - 4*a*c))/(2*a))
        r2 = ((-b - cmath.sqrt(b**2 - 4*a*c))/(2*a))
        r = [r1, r2]
     else:
        n_roots = 1
        r = -b/(2*a)
     return n_roots, r

n_roots, r = my_n_roots(1,0,-9)
print(n_roots, r)
print(my_n_roots(3,4,5))
print(my_n_roots(2,4,2))
```

```
2 [3.0, -3.0]
(-2, [(-0.6666666666666666+1.1055415967851332j), (-0.6666666666666666-1.1055415967851332j)])
(1, -1.0)
```

In [18]:

```
#Problem 3-iteration

def my_n_max(x, n):
    x = sorted(x)
    out1 = max(x1)
    out2  = max(x2)
    out3 = max(x3)
    out = [out1,out2,out3]
    return out
x1 = [7, 3, 10]
x2 = [5, 6, 9]
x3 = [4, 8, 2, 1]
x = [x1,x2,x3]
n = 3
out = my_n_max(x, n)
print(out)
```

```
[10, 9, 8]
```

In [22]:

```
#Problem 4-iteration
from math import *
import numpy as np

def my_trig_odd_even(m):
    r,c = m.shape
    q = np.zeros(m.shape)
    for i in range(r):
        for j in range(c):
            if (m[i,j]%2 == 0):
                q[i, j] = sin(m[i, j])
            else:
                q[i, j] = cos(m[i, j])
    return q

m = np.array([[36, 8, 1], [21, 14, 49]])
my_trig_odd_even(m)
```

Out[22]:

```
array([[-0.99177885,  0.98935825,  0.54030231],
       [-0.54772926,  0.99060736,  0.30059254]])
```

In [34]:

```
#Problem 5-iteration
from math import *
import numpy as np

P = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
Q = np.array([[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]])
def my_mat_mult(P,Q):
     r, p = P.shape
     p, c = Q.shape
     M = np.zeros((r,c))
     for i in range(r):
            for j in range(c):
                summ = 0
                for k in range(p):
                    summ += P[i,k]*Q[k,j]
                M[i,j] = summ
     return M
my_mat_mult(P, Q)
```

Out[34]:

```
array([[30., 30., 30.],
       [70., 70., 70.]])
```

In [44]:

```
#Problem 8-iteration
from math import *
import numpy as np

def roll():
    return np.random.randint(1, 7, size=1)[0]

def my_monopoly_dice():
    running_total = 0
    count = 1
    while 1:
        dice_1 = roll()
        dice_2 = roll()

        running_total += dice_1 + dice_2

        print(f'Roll {count}:  dice_1 = {dice_1}, dice_2 = {dice_2}')
        if dice_1 != dice_2:
            break

        count += 1

    print(f'Running total = {running_total}')
    return running_total


running_total = my_monopoly_dice()
```

```
Roll 1:  dice_1 = 1, dice_2 = 1
Roll 2:  dice_1 = 2, dice_2 = 5
Running total = 9
```

In [47]:

```
#Problem 12-iteration
from math import *
import numpy as np

def my_trig_odd_even(m):
    r,c = m.shape
    q = np.zeros(m.shape)
    for i in range(r):
        for j in range(c):
            if (m[i,j]%2 == 0):
                q[i, j] = cos(np.pi/m[i, j])
            else:
                q[i, j] = sin(np.pi/m[i, j])
    return q

m = np.array([[3, 4], [6, 7]])
my_trig_odd_even(m)
```

Out[47]:

```
array([[0.8660254 , 0.70710678],
       [0.8660254 , 0.43388374]])
```

In [ ]:

In [ ]:

In [ ]: