In [3]:
```
#1
from collections import deque
def Solution(a, b, target):
    m = {}
    cnd = False
    path = []

    q = deque()

    #Initializing with jugs being empty
    q.append((0, 0))

    while (len(q) > 0):

        # Current state
        u = q.popleft()
        if ((u[0], u[1]) in m):
            continue
        if ((u[0] > a or u[1] > b or
            u[0] < 0 or u[1] < 0)):
            continue
        path.append([u[0], u[1]])

        m[(u[0], u[1])] = 1

        if (u[0] == target or u[1] == target):
            cnd = True

            if (u[0] == target):
                if (u[1] != 0):
                    path.append([u[0], 0])
            else:
                if (u[0] != 0):
                    path.append([0, u[1]])

            sz = len(path)
            for i in range(sz):
                print("(", path[i][0], ",",
                    path[i][1], ")")
            break

        q.append([u[0], b]) # Fill Jug2
        q.append([a, u[1]]) # Fill Jug1

        for ap in range(max(a, b) + 1):
            c = u[0] + ap
            d = u[1] - ap

            if (c == a or (d == 0 and d >= 0)):
                q.append([c, d])

            c = u[0] - ap
            d = u[1] + ap

            if ((c == 0 and c >= 0) or d == b):
                q.append([c, d])

        q.append([a, 0])

        q.append([0, b])

    if (not cnd):
```

```
        print("Solution is not possible")

if __name__ == '__main__':

    Jug1, Jug2, target = 4, 3, 2
    print("Path from initial state "
        "to solution state ::")

    Solution(Jug1, Jug2, target)
```

```
Path from initial state to solution state ::
( 0 , 0 )
( 0 , 3 )
( 4 , 0 )
( 4 , 3 )
( 3 , 0 )
( 1 , 3 )
( 3 , 3 )
( 4 , 2 )
( 0 , 2 )
```

In [5]:

```
#2
import itertools, random

deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))

random.shuffle(deck)

print("You got:")
for i in range(7): #change the argument value acc to the number of cards
   print(deck[i][0], "of", deck[i][1])
```

```
You got:
6 of Spade
8 of Heart
2 of Spade
4 of Diamond
9 of Club
7 of Club
6 of Heart
```

In [13]:

```
#3
import random

class Monkey:
    def __init__(self, bananas):
        self.bananas = bananas

    def __repr__(self):
        return "Monkey with %d bananas." % self.bananas

monkeys = [Monkey(random.randint(0, 50)) for i in range(5)]


print ("Random monkeys:")
print (monkeys)

print

def number_of_bananas(monkey):
    """Returns number of bananas that monkey has."""
    return monkey.bananas

print ("number_of_bananas( FIRST MONKEY ): ", number_of_bananas(monkeys[0]))
```

```
print

max_monkey = max(monkeys, key=number_of_bananas)
print ("Max monkey: ", max_monkey)
```

```
Random monkeys:
[Monkey with 20 bananas., Monkey with 3 bananas., Monkey with 35 bananas., Monkey
with 35 bananas., Monkey with 17 bananas.]
number_of_bananas( FIRST MONKEY ):  20
Max monkey:  Monkey with 35 bananas.
```

In [15]:

```
#4
from sys import maxsize
from itertools import permutations
V = 4

def travellingSalesmanProblem(graph, s):

    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)

    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:


        cw = 0

        k = s
        for j in i:
            cw += graph[k][j]
            k = j
        cw += graph[k][s]

        min_path = min(min_path, cw)

    return min_path



if __name__ == "__main__":


    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
             [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

```
80
```

In [ ]: