```
m=0
k=0
for i in range(10): #boucle externe
    m=m+1
    for j in range(10): #boucle interne
        k=k+1
#print('le nombre d iteration de la boucle ####### ',m)
#print('le nombre d iteration de la boucle ####### ',k)


#1 (a)
message=''
n=10
k=3
for i in range(n-k):
    message+=' '#bien mettre un espace !
for i in range(k):
    message+='*'
message+='*'
#print(message)

#1 (b)
message=''
n=10
k=3
for i in range(2*(n-k+1)):
    message+=' '#bien mettre un espace !
for i in range(2*k):
    message+='*'
message+='*'
#print(message)


#1 (c)

## Il y a n=5 lignes ; dans la derniere ligne il y a 9 etoiles = 2*5-1 etoiles.
## on constate aussi que pour la ligne 5 il y a 0 espace, pour la ligne 4 il y a 2 etoiles,
# ligne 3 = 4 etoiles, etc. Soit 2*(n-k-1) espaces pour la kieme ligne et 2k+1 etoiles pour k
                                                    allant de 0 a 4=n-1
def triangle(n):
    for k in range(n):
        message=''
        for i in range(2*(n-k-1)):
            message+=' '
        for i in range(2*k):
            message+='*'
        message+='*'
        print(message)
#print(triangle(15))

## 1(d)

#s=1 : 2 lignes, 3 etoiles
#s=2 : 4 lignes, 7 etoiles
#s=3 : 6 lignes, 11 etoiles
#de maniere generale 2*s lignes, et 4s-1 etoiles

s=2
n=3
for k in range(2*s):
    message=''
    for i in range(2*n-k-1):
        message+=' '
    for i in range(2*k+1):#i va de 1 a 2*(2s-1)+1 = 4s-1
        message+='*'
    #print(message)

#on constate que si on met s=2, on forme bien le bon triangle, mais les
#espaces ne sont pas justes. Pour s=2, il y a forcement 2 espaces de plus dans l'ensemble
# et pour s=1, 4 espaces de plus. En faisant bien les decomptes, les espaces a mettre sont 2n-k-1
```

```
# et non 2s-k-1


## 1(e)
def Sapin(n):
    for s in range(1,n+1):
        for k in range(2*s):
            message=''
            for i in range(2*n-k-1):
                message+=' '
            for i in range(2*k+1):#i va de 1 a 2*(2s-1)+1 = 4s-1
                message+='*'
            print(message)
print(Sapin(3))


#Q2

def premier(n):
    listeNP=[2]
    k=3
    while k<=n:
        test=True
        i=2 #entier de 2 a k
        while i<k  and test==True  :#on verifie que l'on n'a pas trouve de diviseur avant k
            if k%i==0:
                test=False
            i+=1#on passe a l'entier suivant
        if test==True:
            listeNP.append(k)
        k+=1
    return listeNP

#print(premier(100))


#Q3
#On cherche a la main les possibilites, par exemple 50*6 + 4 zero
# ou 50*5+20*2+10 +2*0, etc.

#Q4

def LancerN(x,n):
    L=[]
    for a in range(n+1):
        for b in range(n+1-a):
            for c in range(n+1-a-b):
                if (a+b+c<=n) and a*10+b*20+c*50==x:
                    L.append([a,b,c,a+b+c,a*10+b*20+c*50])
    return L

print(LancerN(300,10))


#Q5
#Dans les faits, si a et b sont fixes, c est forcement fixe d'apres l'equation (1)
#a condition que c soit un entier
def Lancer(x,n):
    L=[]
    for a in range(n+1):
        for b in range(n+1-a):
            c=(x-10*a-20*b)/50
            if int(c)==c and (a+b+c<=n):
                L.append([a,b,int(c),a+b+int(c),a*10+b*20+int(c)*50])
    return L

#print(Lancer(300,100))

from time import time
t1=time()
LancerN(3000,100)
```

```
t2=time()
#print((t2-t1)*1000)
t1=time()
Lancer(3000,100)
t2=time()
#print((t2-t1)*1000)

#on gagne en temps a ne faire que deux boucles imbriquees au lieu de 3,
#le nombre d'operation est d'ordre n**2 au lieu de n**3 en ordre de grandeur
```