Q1

One company sells flowers. Each flower costs 20 SEK. During weeks 14 to 17, they have a sale and offer two flowers for the price of one. In addition, regardless of the week number if a customer buys 200 or more flowers in a single batch he/she gets 20% discount. The two offers cannot be combined and the customer pays according to whichever offer gives a lower price. Implement a program which asks for the number of flowers and the week number and prints the total price. The following are example dialogs with the program:

```
Number of flowers: 10
Week number: 1
The price is 200

Number of flowers: 5
Week number: 15
The price is 60 # 4 flowers for 40 SEK and 1 flower for 20 SEK

Number of flowers: 250
Week number: 20
The price is 4000
```

Answer

```
def flowers():
    flowers = int(input("Number of flowers: "))
    week = int(input("Week number: "))
    price = flowers*20

    price1 = price
    if 14 <= week <= 17:
        price1 = (flowers//2 + flowers%2)*20

    price2 = price
    if flowers >= 200:
        price2 = price2*0.80

    price = min(price1,price2)
    print("The price is "+str(price))

flowers()
```

Q2

The digits sum is the sum of all the digits in a number. For example:

Number	Digits	Sum	Condition
124	$1+2+4$	7	$124 > 12*7$
395	$3+9+5$	17	$395 > 12*17$
4211	$4+2+1+1$	8	$4211 > 12*8$
0	0	0	$0 = 12*0$

Implement the function `find_numbers(n)` which prints all the numbers from 0 to n for which the number is exactly 12 times bigger than its digits sum. Obviously in the table above, 0 is the only number which satisfies the condition.

Hint: Beside zero, there is only one other number for which the condition holds. If you run `find_numbers(1000)` you must be able to see it.

Answer

```
def find_numbers(n):
    for i in range(n):
        s = 0
        j = i
        while j != 0:
            s = s + (j % 10)
            j = j // 10
        if i == 12*s:
            print(i)

find_numbers(1000)
```

```
0
108
```

Q3

In one factory, workers produce 3 types of items and they receive bonuses dependent on how many items of each kind they have produced. The management must therefore keep track of who has produced what. Implement the class ItemsProduction with the following elements:

- An `__init__(bonus1,bonus2,bonus3)` method which initializes the state of the object so that in the beginning no one has produced anything. The

arguments `bonus1, bonus2, bonus3` specify how much bonus each worker receives for each type of item. (2 points)

- A method `reportItem(worker,item)` which takes the name of the worker and the item type and stores the information in the state of the object. (5 points)
- A method `printProduction()` which prints on separate lines the worker's name and the number of items that he/she has produced. (5 points)
- A method `computeBonus(worker)` which takes the name of the worker and returns the bonus that he/she has earned. (5 points)

Answer

```
class ItemsProduction:
    def __init__(self,bonus1,bonus2,bonus3):
        self.bonus1 = bonus1
        self.bonus2 = bonus2
        self.bonus3 = bonus3
        self.production = {}

    def reportItem(self,worker,item):
        self.production.setdefault(worker,[0,0,0])[item] += 1

    def printProduction(self):
        for worker,counts in self.production.items():
            print(worker,counts[0],counts[1],counts[2])

    def computeBonus(self,worker):
        c1,c2,c3 = self.production.get(worker,[0,0,0])
        return c1*self.bonus1 + c2*self.bonus2 + c3*self.bonus3

prod = ItemsProduction(10,20,30)

prod.reportItem("Jane",0)   # Jane produced one item of type 0
prod.reportItem("Karl",1)   # Karl produced one item of type 1
prod.reportItem("Jane",1)   # Jane also produced one item of type 1
prod.reportItem("Kim",2)    # Kim produced one item of type 2
prod.reportItem("Karl",1)   # Karl produced a second item of type 1
prod.printProduction()
print(prod.computeBonus("Jane"))
print(prod.computeBonus("Karl"))
print(prod.computeBonus("Alex"))
```

```
Jane 1 1 0
Karl 0 2 0
Kim 0 0 1
30
40
0
```