Uppgift 1

Följande regler specificerar en adress som kan anges i form av en sträng:

- En adress består av 4 grupper av 3 siffror
- Varje grupp indelas med hjälp av kolon
- Varje grupp består av 3 siffror men ett godtyckligt antal siffror kan utelämnas i gruppen och skall i sådana fall tolkas som inledande nollor.
- Ifall ett eller fler kolon utelämnas skall det tolkas som att det är en utelämnad avslutande grupp som är 000

Följande är exempel på adresser:

Angiven adress:	**Fullständig adress:**
192:168::1	192:168:000:001
00:02:34:567	000:002:034:567
:::	000:000:000:000
:168	000:168:000:000

Ditt jobb är att skapa en funktion som tar emot en angiven adress i form av en sträng och sedan returnerar en sträng med den fullständiga adressen. Skapa sedan ett huvudprogram som testar funktionen och skriver ut resultatet.

Din lösning skall vara rimligt generell. Det är ok att hårdkoda antalet grupper och antalet siffror i varje grupp. Men programmet bör utan större problem kunna utökas för ett godtyckligt antal grupper och siffror i varje grupp, lös alltså inte problemet med fullständig uppräkning.

Körexempel:

```
192:168::1 => 192:168:000:001
00:02:34:567 => 000:002:034:567
::: => 000:000:000:000
:168 => 000:168:000:000
```

Uppgift 2

Du ska sortera en lista genom att implementera en sorteringsalgoritm. Algoritmen är insertion sort och fungerar genom att gå igenom en lista från vänster till höger. Man undersöker sedan varje element i listan parvis med det föregående elementet i listan tills det undersökta elementet placerats tillräckligt långt till vänster. Om det föregående elementet är större än det nuvarande elementet är eller om vi nått början av listan är nuvarande element på rätt plats. Annars behöver nuvarande element byta plats med föregående element och sedan jämföras med föregående element igen tills elementet är på rätt plats. Följande är en beskrivning i pseudokod av algoritmen som sorterar en list i stigande ordning (pilar används för att beskriva tilldelning):

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

Ditt jobb är att skapa en funktion som implementerar denna algoritm för sortering i stigande eller fallande ordning. Sorteringsordningen skall bestämmas genom ett funktionsobjekt (lambda-funktion eller vanlig funktion) som skickas som parameter till funktionen. Funktionen ska inte ha något returvärde. Nedan följer ett körexempel med 2 anrop till funktionen i terminalen, men du får gärna skapa ett huvudprogram som sköter anrop och utskrift av resultaten. Ersätt FUNKTIONSOBJEKT med det funktionsobjekt som behövs för stigande respektive fallande ordning:

Körexempel:

```
l = [2, 1, 7, 6]
>> insertion_sort(l, FUNKTIONSOBJEKT)
>> l
[7, 6, 2, 1]
>> insertion_sort(l, FUNKTIONSOBJEKT)
>> l
[1, 2, 6, 7]
```

Uppgift 3

Collatz problem utgår från följande räknelek:

1. Utgå från ett positivt heltal **n**.
2. Om **n** är jämnt, dela det med två. Om det är udda, multiplicera det med tre och addera därefter ett.
3. Upprepa steg 2 tills **n** blir ett.

Din uppgift är att skapa exakt **2 funktioner** som enskilt utför räkneleken som beskrivs ovan. Den ena funktionen ska implementera räkneleken **iterativt** och den andra skall implmentera räkneleken **rekursivt**. Funktionerna ska skriva ut hur många upprepningar det tog att komma fram till talet 1. Skriv sedan ett huvudprogram som testar båda funktionerna för några olika tal.

Det är inte bevisat att denna algoritm faktiskt gör alla tal till 1 i slutänden. Men åtminstone alla tal upp till 10^20 har testats med datorer och ger resultatet 1 i slutänden. Här är några exempel man kan testa med:

n:	**Antal upprepningar**
1	0
2	1
3	7
4	2
5	5
39	34

Uppgift 4

Ditt jobb är att skapa ett program som hjälper folk hålla koll på skulder. Programmet ska hantera tre olika typer av inmatning:

- Transaktioner matas in på formatet **Givare, Mottagare, Värde**. Hur mycket personer är skyldiga och hur mycket de har innestående ska sparas av programmet fram tills det avslutas
- Utskrift begärs genom att skriva **skriv ut**. Formatteringen av utskriften spelar roll.
- Avslut begärs genom att skriva **avsluta**. När programmet avslutas ska nuvarande skulder skrivas till en fil som heter **debts.txt**

Under tiden som programmet körs skall skulderna lagras i en databehållare av typen **dictionary**.

Körexempel (användarinmatning i fetstil):

```
Skuldprogrammet!
Mata in transaktioner och kommandon:
Pontus, Emma, 20
skriv ut
==================
Nuvarande skulder:
Pontus:        +20
Emma:          -20
==================

Emma, Alex, 25
skriv ut
==================
Nuvarande skulder:
Pontus:        +20
Emma:           +5
Alex:          -25
==================

Alex, Pontus, 15
skriv ut
==================
Nuvarande skulder:
Pontus:         +5
Emma:           +5
Alex:          -10
==================

avsluta
```

Uppgift 5

Din uppgift är att skapa ett program som tar emot 4 argument genom kommandoraden. Argumenten är **slut**, **ord1**, **ord2**. Programmet skall sedan skriva ut alla talen från 1 till och med det angivna slutet. Siffrorna ska skrivas ut i jämna kolumner med 4 siffror per rad (se körexempel (det är viktigt att kolumnerna är jämna, inte exakta antalet mellanslag)). Om talet är jämnt delbart med 3 ska **ord1** skrivas ut istället för siffran. Om talet är jämnt delbart med 4 ska **ord2** skrivas ut istället för siffran. Om talet är jämnt delbart med båda talen ska både **ord1** och **ord2** skrivas ut. Detta problem skall lösas utan loopar och istället bara använda sig av rekursion.

Körexempel:

```
$ python3 uppgift5.py 13 Fizz Bang
1               2               Fizz            Bang
5               Fizz            7               Bang
Fizz            10              11              Fizz Bang
13
```