Del 1 - Du får inte ändra given kod om inte annat anges.

Uppgift 1 – Implementera Algoritm

Skapa en funktion **flatten_list** som tar emot en lista som innehåller godtyckligt många tuplar som i sin tur innehåller godtyckligt många tal. Din uppgift är att "platta till" listan, vilket innebär att vi kopierar alla talen i alla tuplarna till en ny lista. Funktionen ska returnera denna platta lista. Detta ska lösas med följande algoritm:

- Skapa en tom lista
- För varje tupel i listan
  - För varje tal i tupeln
    - Lägg till talet i den nya listan
- Returnera den nya listan

Funktionella krav (användarinmatning i fetstil):

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Uppgift 2 – Abstraktion

Den givna filen **uppgift2.py** innehåller fyra funktioner till ett bokningssystem för en biograf, implementera följande funktionalitet:

create_seating tar emot två heltal **rows** och **seats**, antalet rader och antalet platser per rad. Funktionen ska returnera en lämplig behållare som representerar alla sittplatser i biografen, varje sittplats är i detta skede obokad.

book_seats tar emot en behållare med sittplatser och en lista med sittplatser som ska bokas. Både rad- och sätesnummer börjar från index 0. Om alla platser som ska bokas är lediga så bokas de, annars skrivs ett felmeddelande ut (se köexemplet).

display skall skriva ut er representation av biografen, bokade säten skrivs ut som 1, säten som inte är bokade skrivs ut som 0. Utskriften behöver vara läsbar och korrekt, men formatteringen behöver inte exakt stämma överens med den i de funktionella kraven.

Funktionella krav (endast första 2 raderna presenteras här):

```
Sätet (3, 4) är bokat!
Bokningen misslyckades.
0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
...
```

Uppgift 3 – Behållare

I den givna filen **uppgift3.py** finner du en sträng med vokaler, samt ett par exempelsträngar. Din uppgift är att implementera funktionen **remove_vowels** som tar emot en godtycklig sträng, och plockar bort alla vokaler från strängen, och returnerar en ny sträng med alla vokaler borttagna. För denna uppgift kan du utgå från att alla strängar är på svenska, alla bokstäver är gemener, och att du endast behöver plocka bort de vokaler som finns i den redan existerande variabeln i den givna filen.

Funktionella krav:

```
hj p dg      hr mr d      jg mr br
```

Del 2

Uppgift 4 – Skapa algoritm

I denna uppgift skall du implementera ett enkelt krypterings-skiffer, som fungerar genom att man tar ett alfabet och "förskjuter" det alfabetet X antal tecken åt vänster eller höger för att skapa en dekrypteringstabell. Om vi tänker oss det engelska alfabetet som "a b c d e f . . . u v w x y z", och därefter förskjuter det tre tecken åt höger så får vi istället "x y z a b c . . . r s t u v w". För att kryptera ett meddelande tar man då endast den bokstav som nu ligger på den förskjutna platsen och byter ut det tecknet. Nedan kan du se hur det skulle se ut om strängen "hello world" krypterades med en förskjutning på tre tecken åt höger.

```
hello world
a b c d e f g h i j k l m n o p q r s t u v w q y z
      | |   | |       |     |     |         |
x y z a b c d e f g h i j k l m n o p q r s t u v w
ebiil tloib
```

Din uppgift är att skapa två funktioner - **encrypt** och **decrypt** som tar emot en sträng som skall krypteras eller dekrypteras samt förskjutningen som skall göras på alfabetet. Förskjutningen anges som ett positivt eller negativt heltal. När **encrypt** används ska varje tecken i strängen förskjutas lika många steg åt höger som förskjutningen anger för ett positivt heltal, och åt vänster för ett negativt. **decrypt** fungera på motsatt sätt, alltså vänster för positiva heltal och höger för positiva.

Den givna filen **uppgift4.py** finns en sträng som representerar alfabetet, använd den.

Icke-funktionella krav:

- Det får inte finnas stor koduppprepning mellan **encrypt** och **decrypt**.
- Funktionen skall fungera oavsett definitionen av **alfabet** så länge alla tecken i strängen finns i **alfabet**.

Funktionella krav:

```
Pmttw> &wztl+ :Pwxm )w$'zm pixx) #w !mm um+;
/6DDGb OGJD5S X/GH6 QGM'J6 92HHQ LG K66 E6SY
CZggj& rjmgYv !CjkZ tjp'mZ cVkkt oj nZZ hZv#
,3AADY LDGA2P U,DE3 NDJ'G3 6~EEN ID H33 B3PV
CZggj& rjmgYv !CjkZ tjp'mZ cVkkt oj nZZ hZv#
/6DDGb OGJD5S X/GH6 QGM'J6 92HHQ LG K66 E6SY
Pmttw> &wztl+ :Pwxm )w$'zm pixx) #w !mm um+;
Hello, world! (Hope you're happy to see me!)
```

Uppgift 5 – Abstrakta datatyper

I denna uppgift skall du skapa en ADT som representerar ett lastfartyg. Du väljer helt själv hur du vill modellera din ADT, så länge som all funktionalitet som efterfrågas i beskrivningen nedan finns med.

Ett **lastfartyg** ska ha en maxkapacitet för hur mycket last den kan ta med sig. Denna last kan endast förvaras på vänster eller höger sida av båten.

Last eller gods som lastas på båten kommer alltid att ha en vikt. Den sammanställda vikten av all last får inte överskrida båtens maxkapacitet.

Allt gods som lastas på fartyget skall även ha ett unikt namn.

Det skall finnas funktionalitet för att lasta fartyget med gods, samt att ta bort specifikt gods från båten genom att ange dess namn. När gods lastas måste detta lastas på antingen höger eller vänster sida av båten.

När någonting lastas på båten skall användaren av er ADT ange om det skall lastas på höger eller vänster sida av båten. Last på båten måste vara någolunda balanserad för att fartyget ska kunna lämna hamnen. För att båten ska få lämna hamnen får det endast finnas en 10% skillnad i vikt på höger respektive vänster sida. Till exempel, om höger sida är lastad med 100 kg måste vänster sida vara lastad med mellan 90 - 110 kg.

Det behöver finnas en funktion som kan anropas för att kaptenen skall veta om det är säkert att lämna hamnen. Denna funktion ska returnera **True** om det är säkert att ge sig av (*alltså om lasten är balanserad och maxkapaciteten inte överskridits*) och **False** om det inte är säkert.

Icke-funktionella krav:

- Din ADT skall vara designad på ett sådant sätt att den är enkel att använda, och funktionsnamnen skall vara beskrivande.
- Det skall finnas ett enkelt huvudprogram där du visar funktionaliteten för din ADT. Du ska skapa en båt som kan lämna hamnen och en som inte kan lämna hamnen.

Funktionella krav:

- All funktionalitet i beskrivningen har implementerats.
- All kod som lämnats in fungerar som förväntat utefter beskrivningen.

Uppgift 6 – Referenssemantik

I den givna filen **uppgift6.py** hittar du ett huvudprogram för ett enkelt lagerhanteringsverktyg, där olika förrådsutrymmen lagrar frukter. Förråden representeras av en dictionary, där nyckeln är namnet på förrådet och värdet är en dictionary med frukterna i förrådet. Dictionaryn med frukterna består av nycklar som är namnet på varje frukt som associeras med hur många av den frukten som finns tillgängliga. I den givna filen finns ett antalvarit optimalt funktioner som du behöver implementera följande funktionalitet för:

- **link_warehouses** skall länka samman två förråd så att de delar på sina lager, genom att de har en delad referens till samma dictionary. När två förråd länkas samman skall deras lagersaldon för alla frukter adderas ihop. Har förråd A 20 äpplen och förråd B 10 skall bägge förråd alltså ha 30 äpplen i den delade dictionaryn efter de länkats samman. Du kan utgå från att endast två förråd kan vara länkade på en och samma gång.
- **create_order** tar emot vilket förråd som skall skicka en order, vilken produkt som beställts och hur många produkter som skall inkluderas i ordern. Om det förrådet som skall skicka ordern har länkats ihop med föregående funktion skall bägge de länkade förrådens lagersaldo sänkas lika mycket. Finns det inte tillräckligt av den vara som beställts skall ett enkelt felmeddelande skrivas ut.
- **add_stock** skall lägga till mer av en produkt i ett förråd. Om två förråd har länkats skall deras delade lagersaldo uppdateras.
- **unlink_warehouses** skall bryta länken mellan de två förråden så att de inte längre delar på en referens till samma dictionary. Varje lager får hälften av saldot (godtycklig avrunding).
- **print_warehouses** skall skriva ut lagersaldot för samtliga förråd i en snygg tabell, se körexemplet. Det är okej om det inte är exakt lika många mellanslag i din utskrift som i körexemplet, men stilen skall matcha.

Icke-funktionella krav:

- Din implementation skall fungera oavsett vilka varor som finns i förråden, och att det finns en godtycklig mängd varor i förrådet. Du kan dock alltid utgå ifrån att alla förråd alltid innehåller minst en av varje vara.

Funktionella krav:

```
Ursprungligt lagersaldo:
-----------|------------|------------|------------|------------|
Warehouse  | A          | B          | C          | D          |
-----------|------------|------------|------------|------------|
apple      | 20         | 10         | 30         | 40         |
banana     | 10         | 20         | 20         | 30         |
pear       | 30         | 40         | 50         | 60         |
-----------|------------|------------|------------|------------|

Efter operationerna:
-----------|------------|------------|------------|------------|
Warehouse  | A          | B          | C          | D          |
-----------|------------|------------|------------|------------|
apple      | 10         | 10         | 30         | 40         |
banana     | 15         | 15         | 20         | 30         |
pear       | 40         | 40         | 50         | 60         |
-----------|------------|------------|------------|------------|
```