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

Uppgift 1 – Implementera Algoritm

Utgå från den givna filen **uppgift1.py**. Skapa en funktion **collatz** som tar mot ett heltal (**n**) :

- Fram tills **n** = 1:
  - om **n** är jämnt delbart med 2
    - tilldela **n** resultatet av att dela **n** med 2
  - annars:
    - tilldela **n** resultatet av att multiplicera **n** med 3 och lägg till 1
  - skriv ut **n**

Funktionella krav (användarinmatning i fetstil):

```
n = 1:
n = 2: 1
n = 3: 10 5 16 8 4 2 1
n = 128: 64 32 16 8 4 2 1
```

Fun fact:

Detta är baserat på Collatz problem, där man inte vet om alla **n** blir 1 förr eller senare om man följer denna algoritm. Man har inte kunnat bevisa att det gäller för alla tal, men har med datorer testat alla heltal upp till 2.36×10^{21}

Uppgift 2 – Abstraktion

I den givna filen **uppgift2.py** ska du implementera de deklarerade funktionerna enligt följande beskrivning. Funktionerna kommer tillsammans utgöra grunden för en kö.

- **create_queue()** Skapar och returnerar en tom kö, välj en lämplig behållare.
- **enqueue(queue, element)** Lägger till element sist i kön.
- **dequeue(queue)** Tar bort och returnerar det första elementet i kön. Om kön är tom ska funktionen returnera None och inte ändra på elementen i den underliggande behållaren.
- **is_empty(queue)** Returnerar True om kön är tom, annars False.
- **peek(queue)** Returnerar det första elementet i kön utan att ta bort det. Om kön är tom ska funktionen returnera None.

Funktionella krav (användarinmatning i fetstil):

```
Lägger till A i kön.
Lägger till B i kön.
Lägger till C i kön.
Första element i kön är: A
Tar ut element: A
Första element i kön är: B
Tar ut element: B
Tar ut element: C
Kön är tom!
Tar ut element: None
```

Uppgift 3 – Behållare

I den givna filen **uppgift3.py** finner du några strängar med heltal som avgränsas med blanksteg (mellanslag). Din uppgift är att implementera funktionen **sum_number_string** som returnerar summan av talen i strängen. Du får inte använda den inbyggda funktionen **sum** i denna uppgift.

Funktionella krav:

```
Summan av '1 2 3 4 5' är 15
Summan av '-2 10 -5 7' är 10
Summan av '100 200 300 400 500' är 1500
```

Del 2

Uppgift 4 – Skapa Algoritm

I denna uppgift skall du lösa ett problem i två delar. I den givna filen **uppgift4.py** finns en lista av ett jämnt antal unika heltal. Sortera denna lista. Det är okej att skapa in ny lista som är sorterad istället för att ändra i den ursprungliga listan.

När listan är sorterad ska du dela listan i två på mitten, vi kallar dessa del a och del b:

```
[1, 3, 4, 5, 7, 8, 10, 11]
            ^
            |
         mitten

Del a => [1, 3, 4, 5]
Del b => [7, 8, 10, 11]
```

Skapa sedan en ny lista som innehåller par av tal. Varje tal i del a ska paras ihop med ett tal i del b. Talet som väljs i del b ska vara det vars summa tillsammans med talet från del a som är närmast ett målvärde, i detta fall 12.

Icke-funktionella krav:

- Du får inte använda pythons inbyggda funktioner för sortering (så som **list.sort()**). Du behöver alltså implementera din egen sortering av listan (det behöver inte vara en snabb algoritm). Du får använda andra inbyggda funktioner som stöd när du implementerar sorteringen (så som **min(a, b)** och **max(a, b)**.
- Implementationen ska vara generell nog för att fungera för en godtycklig lista av unika heltal som har en jämn längd.

Funktionella krav:

```
[[1, 11], [3, 8], [4, 8], [5, 7]]
```

Uppgift 5 – Abstrakta datatyper

I denna uppgift ska du skapa en ADT som representerar ett spelbräde. Du väljer själv hur du vill modellera din ADT, men all funktionalitet som efterfrågas i beskrivningen nedan skall finnas med. Ett spelbräde består av ett antal rutor, antalet rutor bestäms av bredden och höjden på spelbrädet. Spelbrädets bredd och höjd anges när spelbrädet skapas.
På varje ruta kan det stå ett godtyckligt antal spelpjäser (vi skulle även kunna modellera spelpjäser med en ADT, men i denna uppgift antar vi att en spelpjäs är ett valfritt tecken). Programmeraren som kommer arbeta med detta spelbräde har begärt följande funktionalitet:

- Skapa ett spelbräde.
- Placera en pjäs på valfri plats på spelbrädet.
- Ta bort en pjäs på valfri plats på spelbrädet.
- Flytta en pjäs från en plats till en annan. Om pjäsen inte finns på ursprungsplatsen ska inget hända.
- Ta reda på vilka pjäser som står på en plats på spelbrädet
- Visa upp (skriva ut spelbrädet) i valfritt format där det är lätt att se var pjäserna står. Vid utskrift skrivs den senast tillagda pjäsen ut på en plats om det finns fler än en.

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 ett spelbräde, ställa dit pjäser (minst två på en plats), skriva ut spelbrädet och flytta en pjäs.

Funktionella krav:

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

Uppgift 6 – Referenssemantik

Pontus driver en rollspelskampanj där spelarnas karaktärer har en speciell typ av väska som kan användas för att dela magiska ”potions” med varandra. Du ska implementera funktionerna nedan för att hjälpa Pontus spåra antalet potions spelarna har.

I filen **uppgift6.py** finns en dictionary med alla spelare i gruppen. Spelarnas namn är nycklar som associeras med en dictionary som i sin tur representerar vilka potions varje spelare har. I uppgiften kommer du implementera funktioner som länkar ihop och länkar isär två spelares inventarier.

Du ska implementera följande funktioner i filen **uppgift6.py**:

- **link_characters(characters, name1, name2)**
  Länkar ihop två karaktärer så att de delar på samma inventarielista, denna delning ska ske genom en referens. När de länkas slås antalet av föremålen ihop. Anta att karaktärerna som länkas samman inte redan är länkade med andra karaktärer.

- **add_item(characters, name, item, amount)**
  Lägger till ett visst antal av ett föremål i en karaktärs inventarie. Är karaktären länkad uppdateras båda.

- **use_item(characters, name, item, amount)**
  Använder ett visst antal av ett föremål för en karaktär. Är karaktären länkad minskas bådas saldo. Du behöver inte ta hänsyn till att fler föremål skulle användas än vad en karaktär har.

- **unlink_characters(characters, name1, name2)**
  Bryter länken mellan två karaktärer. När länken bryts delas inventariet i två lika stora delar (avrundning görs godtyckligt). Anta att karaktärerna som skickas in är länkade med varandra.

- **print_inventories(characters)**
  Skriver ut inventarierna för samtliga karaktärer i tabellform. Exakt formattering spelar ingen roll men det ska vara lätt att läsa.

Funktionella krav:

```
Ursprungligt läge:
Character    |     Alice|       Bob|     Clara|
-------------|----------|----------|----------|
HealthPotion |         3|         1|         5|
ShieldPotion |         0|         0|         1|
SpeedPotion  |         2|         0|         2|
-------------|----------|----------|----------|

Efter operationer:
Character    |     Alice|       Bob|     Clara|
-------------|----------|----------|----------|
HealthPotion |         2|         2|         5|
ShieldPotion |         0|         0|         1|
SpeedPotion  |         0|         0|         2|
-------------|----------|----------|----------|
```