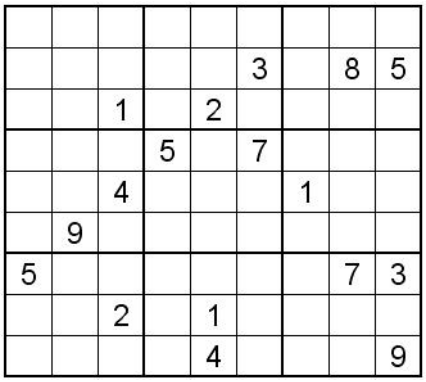
Uppgift 1



Dumbledore gillar Sudoku som är ett pusselspel där man ska fylla i ett 9x9 stort bräde med siffrorna 1 till 9. Siffrorna får bara förekomma en gång per rad, per kolumn samt per mindre 3x3 ruta.

På filen sudoku.txt finns en sudoku-plan. Tomma platser är markerade med 0. Din uppgift är att skapa en ADT som ska representera denna sudoku-plan. Du ska skapa 3 funktioner till din ADT:

- Den första ska läsa in sudoku-planen från filen. Och returnera en abstrakt datatyp.
- Den andra ska hämta ut ett visst värde från en vald plats på planen. Om man försöker hämta ett värde som ligger utanför spelplanen skall funktionen returnera ***-1***. Du kan anta att spelplanen alltid har 9x9 rutor.
- Den sista ska sätta ut ett värde på en vald plats på planen. Är positionen utanför planen ska ingenting ske. Om värdet som sätts inte är mellan 1-9 ska ingenting ske.

När du har implementerat dina adt-funktioner korrekt så ska det givna huvudprogrammet i *uppgift1.py* fungera enligt körexemplen nedan:

Körexempel 1:
Siffrans rad: **1**
Siffrans kolumn: **2**
Siffran på denna position är 0

Positionens rad: **1**
Positionens kolumn: **2**
Vilket värde vill du sätta positionen? **1**
Siffran på denna position är 1

Körexempel 2:
Siffrans rad: **10**
Siffrans kolumn: **8**
Siffran på denna position är -1

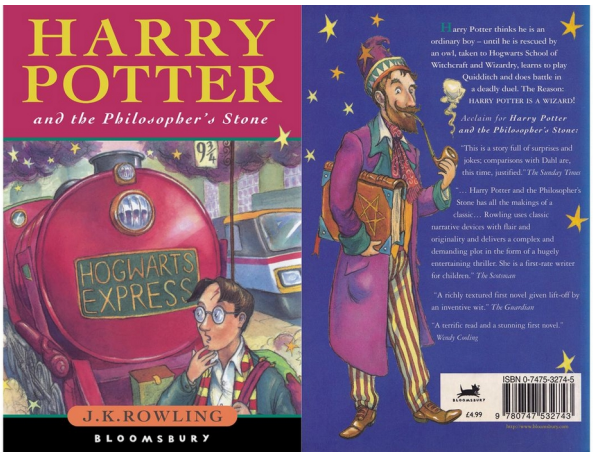
Positionens rad: **8**
Positionens kolumn: **-1**
Vilket värde vill du sätta positionen? **5**
Siffran på denna position är -1

Körexempel 3:
Siffrans rad: **9**
Siffrans kolumn: **9**
Siffran på denna position är 5

Positionens rad: **3**
Positionens kolumn: **8**
Vilket värde vill du sätta positionen? **4**
Siffran på denna position är 4

Uppgift 2

McGonagall gillar inte aktier men förstår att det är klokt att investera sina pengar på ett bra sätt. Hon köper därför böcker som hon tror kommer öka i värde. För att projicera sin avkastning ber hon dig skriva ett program där man får mata in värdet av en bok vid inköp, antalet år man tänker behålla boken, förväntad ökning av värdet i procent (för varje år), samt med vilket intervall man vill se avkastningen. Ditt program ska sedan skriva ut ett antal rader där året, värdet på boken det året, samt värdeökningen hittills framgår. Utskriften skall vara formatterad enligt körexemplen nedan.



Krav: Utskriften av en rad ska skötas av en funktion som tar 3 parametrar (nuvarande år, nuvarande värde, samt startvärdet)

Krav: Alla värden (värdet på boken och ökningen) skrivs ut avrundade till närmaste heltal. Denna avrunding skall göras för utskriften men inte påverka värdet vid beräkning.

Förtydligande: Sista året som ska projiceras kan utelämnas om steglängden för utskrift inte går jämnt ut, startåret (2021) ska dock alltid finnas med

Körexempel 1 (Användarinmatning är i fetstil):

```
Mata in värde vid inköp: 100
Mata in hur många år du vill projicera värdet: 40
Mata in förväntad värdeökning(i procent): 10
Mata in steglängd för utskrift: 5
  År:   Värde:  Värdeökning:
============================
 2021    100kr            0%
 2026    161kr           61%
 2031    259kr          159%
 2036    418kr          318%
 2041    673kr          573%
 2046   1083kr          983%
 2051   1745kr         1645%
 2056   2810kr         2710%
 2061   4526kr         4426%
```

Körexempel 2 (Användarinmatning är i fetstil):

```
Mata in värde vid inköp: 1000
Mata in hur många år du vill projicera värdet: 39
Mata in förväntad värdeökning(i procent): 4
Mata in steglängd för utskrift: 5
  År:   Värde:  Värdeökning:
============================
 2021   1000kr            0%
 2026   1217kr           22%
 2031   1480kr           48%
 2036   1801kr           80%
 2041   2191kr          119%
 2046   2666kr          167%
 2051   3243kr          224%
 2056   3946kr          295%
```

Uppgift 3

Hermione gillar kluriga taktiska spel och Dam är en av hennes favoriter. I den givna filen *uppgift3.py* finns lite given kod som anger siffrorna och bokstäverna för raderna, respektive kolumnerna för ett damspelbräde. Det finns också 2 listor som indikerar vilka rutor det skall finnas vita, respektive svarta pjäser på.

Ditt jobb är att utifrån denna givna data skapa en databehållare som representerar spelbrädet på ett lämpligt sätt. Programmet ska sedan skriva ut spelbrädet och ge användaren möjligheten att mata in drag för att flytta en pjäs. Du behöver inte göra några rimlighetskontroller av inmatningen eller ta hänsyn till några regler vid förflyttningen, användaren vet hur man spelar spelet. När en föflyttning är klar så ska spelbrädet skrivas ut igen och man ska få en ny chans att mata in, och så vidare. Föreslagen lösning i pseudokod finns i den givna filen *uppgift3.py*.

Tips: Unicode-tecken för pjäserna är \u26c2 respektive \u26c0, man kan alltså skriva ut en svart pjäs genom att skriva *print(”\u26c2”)*

Tips: Genom att skapa en dictionary där alla rutor på spelplanen är värden och dess koordinater är nycklar gör det enkelt att komma åt rutan man letar efter. Nyckel *”a3”* skulle alltså kunna leda till strängen *”\u26c2”* (⛂) och nyckeln ”a5” skulle kunna leda till strängen ” ” (sträng av mellanslag).

Tips: Utskriften behöver **inte** se ut exakt som körexemplet, men funktionaliteten ska vara densamma (Se ”Körexempel enklare utskrift” nedan)

Körexempel 1 (användarinmatning är i fetstil):

```
  a  b  c  d  e  f  g  h
1 ⛂     ⛂     ⛂     ⛂
2    ⛂     ⛂     ⛂     ⛂
3 ⛂     ⛂     ⛂     ⛂
4
5
6    ⛀     ⛀     ⛀     ⛀
7 ⛀     ⛀     ⛀     ⛀
8    ⛀     ⛀     ⛀     ⛀
Från: e3
Till: f4
  a  b  c  d  e  f  g  h
1 ⛂     ⛂     ⛂     ⛂
2    ⛂     ⛂     ⛂     ⛂
3 ⛂     ⛂           ⛂
4                ⛂
5
6    ⛀     ⛀     ⛀     ⛀
7 ⛀     ⛀     ⛀     ⛀
8    ⛀     ⛀     ⛀     ⛀
Från:
```

Körexempel enklare utskrift (:

```
⛂⛂⛂⛂

 ⛂⛂⛂⛂

⛂⛂⛂⛂




 ⛀⛀⛀⛀

⛀⛀⛀⛀

 ⛀⛀⛀⛀

Från:
```

Körexempel 2 (användarinmatning är i fetstil):

```
  a  b  c  d  e  f  g  h
1 ⛂     ⛂     ⛂     ⛂
2    ⛂     ⛂     ⛂     ⛂
3 ⛂     ⛂     ⛂     ⛂
4
5
6    ⛀     ⛀     ⛀     ⛀
7 ⛀     ⛀     ⛀     ⛀
8    ⛀     ⛀     ⛀     ⛀
Från: e3
Till: d4
  a  b  c  d  e  f  g  h
1 ⛂     ⛂     ⛂     ⛂
2    ⛂     ⛂     ⛂     ⛂
3 ⛂     ⛂           ⛂
4           ⛂
5
6    ⛀     ⛀     ⛀     ⛀
7 ⛀     ⛀     ⛀     ⛀
8    ⛀     ⛀     ⛀     ⛀
Från: d6
Till: d4
  a  b  c  d  e  f  g  h
1 ⛂     ⛂     ⛂     ⛂
2    ⛂     ⛂     ⛂     ⛂
3 ⛂     ⛂           ⛂
4           ⛀
5
6    ⛀           ⛀     ⛀
7 ⛀     ⛀     ⛀     ⛀
8    ⛀     ⛀     ⛀     ⛀
Från:
```

Uppgift 4

Lockhart har inte tid för triviala saker när han programmerar. Nu ska han skapa ett program som i början kräver att användaren matar in argument via kommandoraden. Han tänker inte besvära sig med att validera vad användaren matar in utan lämnar det till sin elev (dig).

Ditt jobb är att implementera en funktion så att den givna koden i *uppgift4.py* fungerar som förväntat. Funktionen tar 2 parametrar: kommandoradsargumenten, samt en dictionary som specificerar regler kring argumenten. Reglerna specificerar dels hur många argument användaren matar in samt om typen är ”*string*” eller ”*integer*”. Om argumentet ska vara av typen ”*integer*” så får man också, om man vill, specificera max och min värde för den typen. Om man inte vill specificera max och/eller min, så utelämnas det nyckel-värde-paret (se givna filen).

Om valideringen lyckas ska underprogrammet returnera 0 och en tom lista. Om fel antal argument hittas ska underprogrammet returnera 1 och en lista med en sträng som indikerar att det är fel antal argument. Om antalet argument är korrekt men andra fel hittas ska underprogrammet returnera 1 och en lista över felen

Krav: Kontrollen av om ett argument är ett heltal skall skötas med pythons felhantering. Felet skall hanteras inom funktionen *validate_args*.

Tips: Första argumentet till ett program (argument 0) är namnet på filen som körs. Detta argument ska inte valideras av funktionen och finns inte med i dictionaryn som specificerar valideringskriterierna.

Körexempel 1:
```
python3 uppgift4.py 24 "en sträng" 400
(0, [])
```
Körexempel 2:
```
python3 uppgift4.py 24 "en sträng" 400 "extra"
(1, ['Wrong number of arguments'])
```
Körexempel 3:
```
python3 uppgift4.py -20 "en sträng" 550
(1, ['Argument: 1, is smaller than min: 0',
'Argument: 3, is bigger than max: 500'])
```

Körexempel 4:
```
python3 uppgift4.py -20 "en sträng" 550 "extra"
(1, ['Wrong number of arguments'])
```

Uppgift 5

Herr Potter har tråkigt och vill att du implementerar ett spel. I spelet utgår man från en vald siffra och följer några simpla regler för att ändra siffran. Spelet slutar när siffran antingen är 1 eller blir mindre än 1. Blir talet du får i slutändan 1 vinner datorn, blir det mindre än 1 vinner du. **Detta spel skall implementeras rekursivt.**
Reglerna är som följer:

- Är talet jämnt ska du halvera det
- Är talet udda multipliceras det med 3 och ökar sedan med 1 ($N = 3N + 1$)
- Återupprepa dessa steg tills du har ett svar

Den rekursiva funktionen skall steg för steg skriva ut värdet på talet (se körexemplet). Vinner du ska ”Du vann!” skrivas ut. Vinner datorn ska ”Jag vann!” skrivas ut.

Tips: För att kolla om en siffra är jämn eller udda kan du använda dig av modulus-operatorn, %.

Kommentar: Man tror idag att just denna algoritm aldrig kan leda till 0 (Om den angivna siffran är större än 0) utan alltid blir 1. **Försök alltså inte hitta ett testfall som leder till att du vinner**. Däremot är det viktigt att du tar hänsyn till att fallet kan uppstå i din kod eftersom det inte är bevisat att slutresultatet blir 1 för alla tal större än 1.

Körexempel 1: Välj en siffra: **16** 16 -> 8 -> 4 -> 2 -> 1 -> Jag vann! **Körexempel 2:** Välj en siffra: **21** 21 -> 64 -> 32 -> 16 -> 8 -> 4 -> 2 -> 1 -> Jag vann!	**Körexempel 3:** Välj en siffra: **3** 3 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1 -> Jag vann!