Inlämningsuppgift 3:
Simulering av en swingtradingstrategi för aktiehandel

Bakgrund

Att utnyttja den naturliga pendlingen upp och ner i priset på en aktie, genom att köpa och äga aktier endast några dagar i taget, kallas i tradingsammanhang för *swingtrading*. Det är välkänt att aktier har en tendens att uppvisa överdriven volatilitet och att en nedgång oftare följs av en uppgång än ytterligare en nedgång. I boken ”Framgångsrik aktiehandel – 10 vinnande strategier” presenterar Peter Nilsson, Johnny Torssell och Johan Hellström bland annat en swingtradingstrategi vid namn *MagicK*. Den går, enkelt förklarat, ut på att köpa aktier som ser ut att ha nått botten av en överdriven nedåtpendling strax innan börsen stänger för dagen, för att sedan sälja igen vid stängning några dagar senare när aktien pendlat tillbaka upp.

I denna inlämningsuppgift ska du skriva **ett modulärt program** som **läser in historisk data** för en aktie och **simulerar swingtrading med MagicK-strategin**. Syftet är att fastställa om strategin har en *edge* (d v s om den leder till fler/större vinstaffärer än förlustaffärer och därmed har potential att öka en investerares kapital). Din kod ska **beräkna avkastningen och längden** (antal dagar) för en genomsnittlig affär. Den ska även **rita upp två figurfönster**, ett som visar fördelningen av vinster/förluster och ett som visar hur aktiekursen och den signal som tradern använder varierar under de sista 30 dagarna i den inlästa datan. Den text som **skrivs ut till Command Window** ska även **skrivas till en fil**.

Mål

Inlämningsuppgiften avser att testa:

1. Algoritmer, variabelhantering, loopar, logiska satser
2. Skapa och anropa funktioner
3. Inläsning av data från csv-fil
4. Formaterad utskrift till fil
5. Hantering av strukturer och cellmatriser
6. Att utifrån en uppgiftsbeskrivning kunna skriva ett mindre program

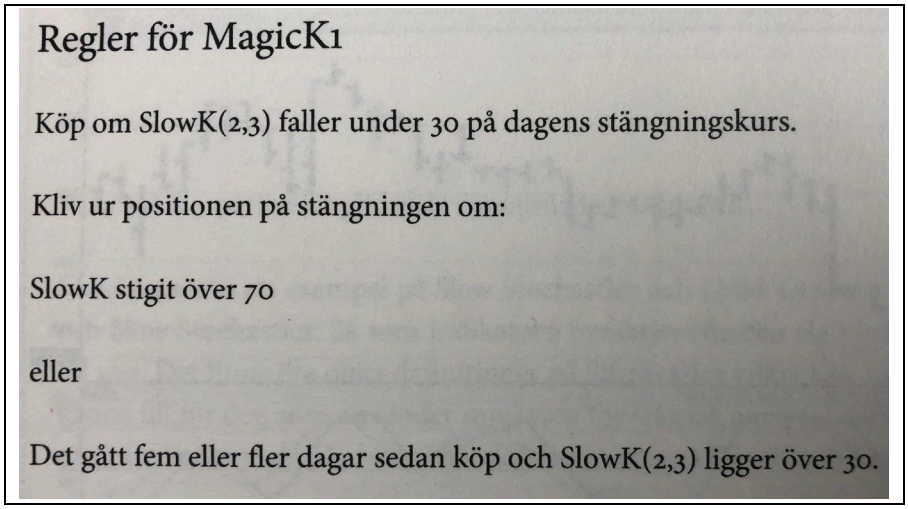
Algoritm

I denna uppgift ska vi skriva ett program som simulerar MagicK1-strategin för aktiehandel[1]. Vad man ska göra i denna strategi styrs av värdet på en variabel, *SlowK(2,3)*, som är ett *tre dagars glidande medelvärde* av en annan variabel, *FastK(2)*. Med andra ord summerar man de tre senaste handelsdagarnas *FastK(2)* och dividerar med tre för att få *SlowK(2,3)*. *FastK(2)* beräknas i sin tur så här för en enskild handelsdag:

$$FastK(2) = \frac{100 * (stängningskursen - lägsta\ noteringen\ de\ senaste\ 2\ dagarna)}{högsta\ noteringen\ de\ senaste\ 2\ dagarna - lägsta\ noteringen\ de\ senaste\ 2\ dagarna}$$

FastK(2) och *SlowK(2,3)* blir med dessa definitioner tal som varierar mellan 0 och 100, och vi har alltså ett värde på *SlowK(2,3)* och ett värde på *FastK(2)* per dag.

Reglerna för MagicK1 är sedan enkla:



[1] Källa: Kapitel 4 i ”Framgångsrik aktiehandel – 10 vinnande strategier” av Nilsson, Torssell & Hellström (Aktiespararna Kunskap, ISBN 978-91-89212-43-5). Det finns flera olika varianter av MagicK, här använder vi oss av MagicK1.

Figur 1 nedan illustrerar hur en aktiekurs kan variera under en handelsdag på Stockholmsbörsen, från öppning 09:00 till stängning 17:25 (i detta exempel Hennes & Mauritz B från 2/10-2019).

Figur 1: Illustration över variationerna i pris för en Hennes & Mauritz B-aktie under en dålig dag på Stockholmsbörsen 2019 (onsdag 2/10). Aktien öppnade på 189,62 kr (vilket visade sig bli dagens högsta) och stängde på 184,44 kr, med en lägstanotering på 182,92 kr. Bilden är ett screenshot från nätmäklaren Nordnets app.

I vår simulering kommer vi att anta att vi går in i en affär (köper aktier) vid stängning klockan 17:25 om *SlowK(2,3)* faller under 30, och stannar tills *SlowK(2,3)* är över 70 eller att det har gått fem dagar eller mer och *SlowK(2,3)* i alla fall är tillbaka över 30. Vi har alltså bara en pågående affär vid varje givet tillfälle.

För att inte komplicera uppgiften onödigt mycket kommer vi att anta att vi inte har några omkostnader för aktiehandeln[2]. Vi kan därmed ”räkna på 1 aktie”, d v s anta att i varje affär så köper eller säljer vi en enstaka aktie.

[2] I verkligheten behöver man ofta betala någon form av *courtage*, d v s en avgift för själva köpet eller försäljningen av aktier.

Uppgift

Börja med att ladda ner de fyra datafilerna (`ERIC_B-2015-01-04-2019-01-04.csv`, `HM_B-2015-10-01-2019-10-01.csv`, `TELIA-2012-01-25-2019-07-02.csv` och `VOLV_B-2012-07-03-2019-07-04.csv`)[3]. Du kan använda dig av en enskild datafil när du utvecklar ditt program, men testa därefter att ditt program fungerar även med de andra datafilerna (detta är ett enkelt test för att se till att du inte hårdkodat in någon information). Filerna finns att hämta i mappen ”Inlämningsuppgift 3” bland kursens dokument i Canvas.

Skriv ett program, i form av en skriptfil `MagicK1.m` i Matlab, som:

- **Läser in data från en av datafilerna** och sparar den data du behöver i en *struct*. Detta ska åstadkommas av en funktion, `readStockData`, som du också ska skriva (mer information om denna senare).

- Efter inläsningen av data behöver programmet **simulera swingtrading enligt MagicK1-strategin**. Detta ska åstadkommas av en funktion, `simulateStrategy`, som du också ska skriva (mer information om denna senare). Denna funktion ska använda den *struct* som den första funktionen returnerade som invariabel, och ska svara med en annan *struct* som innehåller en sammanställning över samtliga affärer som genomförts.

- Programmet ska därefter **skriva ut resultatet av simuleringen till skärmen** och **rita upp de efterfrågade graferna**. Även detta ska göras av en funktion, `printAndPlotResult`, som du också ska skriva (mer information om denna senare). Med andra ord kommer själva huvudprogrammet `MagicK1.m` att vara kort och se ut ungefär på följande sätt:

[3] Du kan själv hämta data för andra aktier och/eller perioder här: http://www.nasdaqomxnordic.com/aktier I denna uppgift har vi medvetet valt en tidsperiod som inte inkluderar Corona-raset, vilket föregicks av en ökad volatilitet som borde signalera ökad vaksamhet. Se extramaterialet om algoritmiskt tänkande för en aktiehandeluppgift specifikt om Corona-raset!

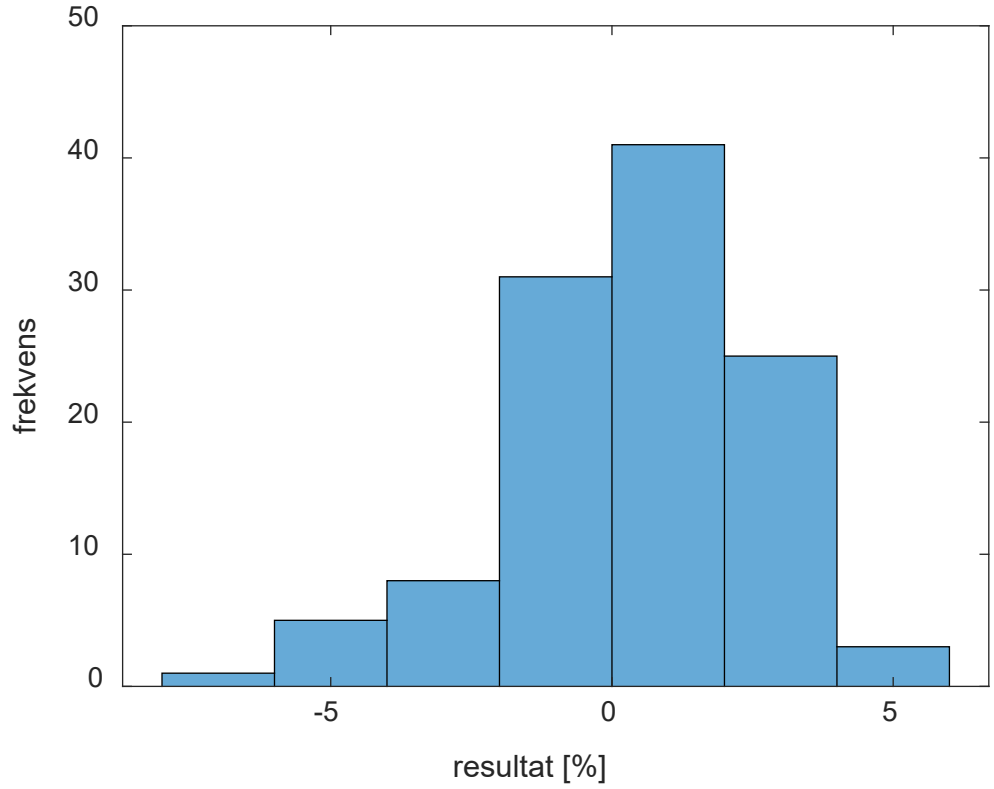
```
clear all
close all
clc

% Read data
filename = 'TELIA-2012-01-25-2019-07-02.csv';
stock = readStockData(filename);

% Simulate strategy
transaction = simulateStrategy(stock);

% Print and plot results
printAndPlotResult(stock, transaction);
```

- Utskriften till kommandofönstret ska se ut på följande sätt (exempel med `TELIA-2012-01-25-2019-07-02.csv` som indatafil):

```
Totalt antal dagar i data: 1862
Antal MagicK-lägen: 115 (6.18%)
Genomsnittligt utfall: 0.38%
Genomsnittlig träffsäkerhet: 59%
Genomsnittligt antal dagar i affär: 5.01
```

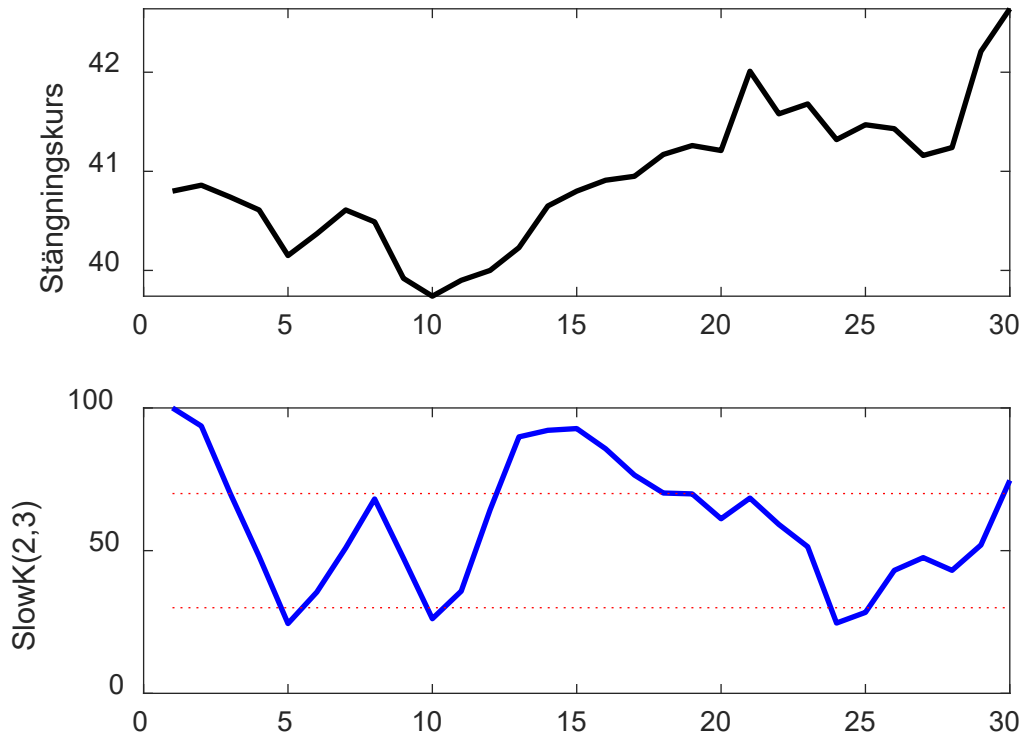
  Exakt samma text ska också skrivas till en textfil, `myResult.txt`.

- Den första av de två figurerna som ska ritas upp är ett histogram över utfallet i de affärer som har genomförts:



Det exakta utseendet på grafen är inte viktigt denna gång, men det ska vara ett histogram som visar hur många affärer (frekvens) som faller inom ett givet spann i termer av utfall (resultat) i procent.

- Den andra figuren ska visa hur aktiekursen utvecklades under *de 30 sista dagarna i den inlästa datan* i ett delfönster, och hur *SlowK(2,3)* varierar i ett annat delfönster (placerat under det första). Återigen är inte det exakta utseendet på grafen viktigt, utan fokus är på innehållet. Ett exempel nedan visar hur det kan se ut med `TELIA-2012-01-25-2019-07-02.csv` som indatafil:



Rent generellt gäller för denna uppgift att det är *hur programmet fungerar* som är det viktiga, inte exakt hur du löst uppgiften eller exakt hur graferna ser ut (bara informationen är den som efterfrågas och i övrigt korrekt). I denna anda följer nu information om hur de tre funktionerna kan konstrueras. *Om du löser uppgiften på ett annat sätt så är det helt OK, så länge din lösning fungerar och inte bygger på någon form av hårdkodning!*

Funktioner

Funktionen `readStockData` ska ha ett inargument och ett utargument.

Inargument: `filename` en textsträng (char-vektor eller string) som anger filnamnet på den datafil som ska användas

Utargument: `stock` en structure som innehåller fem fält: `high`, `low`, `close`, `FastK` och `SlowK`

Funktionen ska läsa in data från filen på valfritt sätt (använd t ex `importdata`, *Import Data*-guiden med *Generate script/function*, eller vad du nu känner dig mest bekväm med). Bilden nedan visar hur början på en datafil ser ut om vi kikar på den i Excel:

	A	B	C	D	E	F	G	H	I	J	K
1	Date	Bid	Ask	Opening price	High price	Low price	Closing price	Average price	Total volume	Turnover	Trades
2	2019-07-02	42,5	42,53	42,25	42,74	42,12	42,64	42,538	10719375	455975365,3	6959
3	2019-07-01	42,19	42,2	41,65	42,32	41,59	42,21	42,086	13346907	561691322,2	8339
4	2019-06-28	41,32	41,33	41,23	41,51	41,07	41,24	41,284	10681011	440954465,5	5473
5	2019-06-27	41,07	41,1	41,38	41,4	41,05	41,16	41,163	7209163	296754887,4	4230
6	2019-06-26	41,3	41,32	41,51	41,52	41,26	41,43	41,39	6047530	250308858,3	3747
7	2019-06-25	41,53	41,55	41,32	41,6	41,12	41,47	41,457	6546448	271383110,6	3877
8	2019-06-24	41,24	41,26	41,44	41,76	41,24	41,32	41,419	10807744	447606234	6176
9	2019-06-20	41,63	41,65	41,92	42,04	41,59	41,68	41,705	14829120	618445462,1	5740
10	2019-06-19	41,64	41,67	42,07	42,07	41,58	41,58	41,692	10459539	436092724,4	4745
11	2019-06-18	42,03	42,04	41,32	42,1	41,18	42,01	41,812	12695128	530855575,1	5270
12	2019-06-17	41,24	41,26	41,35	41,53	41,21	41,21	41,319	5829753	240874365	3694

Vi är intresserade av att plocka ut High price (högstakurs, till fältet `high`), Low price (lägstakurs, till fältet `low`) och Closing price (stängningskurs, till fältet `close`).

Att datafilerna innehåller ,-tecken istället för . som decimalavskiljare kan, beroende på hur du läser in datan, ställa till problem. Du kan t ex använda `regexp` för att byta ’`,`’ mot ’`.`’ i en teckensträng.

Datafilerna har hämtats direkt från Stockholmsbörsen och är sorterade *i fallande datumordning*. För att simuleringen ska bli logisk behöver du därför byta ordning (”flippa”) vektorerna som sparats i de olika fälten. Detta kan du göra t ex med `fliplr`.

Vi vill också fylla fälten `FastK` och `SlowK` genom att utnyttja de formler som tidigare angivits. Notera att FastK inte kan definieras för första dagen i data (det behövs två dagar enligt den angivna formeln), så för första dagen kan du sätta FastK = 100*(close – low)/(high – low). På samma sätt kan inte SlowK definieras förrän för den tredje dagen i data (det behövs tre dagar för att ta fram ett tre dagars medelvärde), så för första dagen kan du sätta SlowK = FastK, och för andra dagen SlowK = (FastK[dag 1] + FastK[dag 2])/2. Tänk på att du måste ha flippat datumordningen innan du beräknar FastK och SlowK!

Funktionen `simulateStrategy` ska ha ett inargument och ett utargument.

Inargument: `stock` utargumentet från `readStockData`

Utargument: `transaction` en vektor av structures där varje element i struct-vektorn representerar en affär; structen har fem fält: `buy`, `sell`, `days`, `gain` och `active`

Vi bör börja med att markera att vi är redo att göra vår första affär, och att vi ännu inte äger några aktier:

```
j = 1;
transaction(j).active = false;
```

Funktionen behöver därefter loopa över alla dagar (`i`) som ingår i datan i `stock` och för varje dag avgöra huruvida något av följande är uppfyllt:

1) Vi äger inte aktien (`transaction(j).active` är `false`) och `SlowK(i) < 30`. Om detta är fallet ska en ny affär initieras: `transaction(j).active` ska bli `true`. I `transaction(j).buy`-fältet ska stängningskursen sparas (från `close`-fältet i `stock`), vilket är den kurs som man köper aktier för denna gång. Index (`i`) för aktuell dag i `stock` ska sparas i `days`-fältet, så att vi kan hålla koll på hur många dagar vi ägt aktierna.
2) Om vi äger aktien (`transaction(j).active` är `true`) och `SlowK(i) > 70` *eller* det har gått fem handelsdagar eller mer sen vi köpte aktien och `SlowK(i) > 30`. Om detta är fallet ska vi sälja aktien: `transaction(j).active` ska bli `false`. I `sell`-fältet ska stängningskursen sparas (från `close`-fältet i `stock`), vilket är den kurs som man säljer aktien för i denna affär. `days`-fältet ska uppdateras så att det nu sparar hur många dagar vi ägde aktien. Dessutom ska `gain`-fältet fyllas med information enligt följande:

```
transaction(j).gain = (transaction(j).sell - transaction(j).buy)/transaction(j).buy;
```

där `j` är den aktuella affären. Sist men inte minst behöver vi öka på `j` med 1 och därefter sätta

```
transaction(j).active = false;
```

Detta markerar att vi är redo för nästa affär, och att vi ännu inte äger några aktier.

Funktionen `printAndPlotResult` ska ha två inargument och inga utargument.

Inargument: `stock` utargumentet från `readStockData`
`transaction` utargumentet från `simulateStrategy`

Funktionen ska skriva ut ett textmeddelande likt det som visas i exemplet på sidan 5, dels till Command Window och dels till en fil, `myResult.txt`. Utskriften görs lämpligen med `fprintf` och kan då skrivas på samma sätt till skärm och fil (m h a `fopen` och `fclose`). Histogrammet ritas med `histogram` och stängningskurs och SlowK för de sista 30 dagarna med `plot` i ett delat figurfönster med `subplot`. Det är informationen i vektorn `[transaction.gain]` som ska illustreras i histogrammet och informationen i slutet av vektorerna `stock.close` och `stock.SlowK` som ska illustreras i linjegraferna.

Totalt antal dagar i datan kan erhållas ur dimensionerna på `stock`. Antalet MagicK1-lägen utläses från längden på `transaction`, och kan sättas i förhållande till storleken på `stock`. Det genomsnittliga utfallet är ett enkelt medelvärde av `[transaction.gain]` och den genomsnittliga träffsäkerheten avgörs av antalet affärer med `transaction.gain > 0` i förhållande till det totala antalet affärer. Det genomsnittliga antalet dagar i affär är ett enkelt medelvärde av `[transaction.days]`.

Ledning

Det är som vanligt **starkt rekommenderat att dela in uppgiften i mindre delar** för att varje del för sig ska bli mer överskådlig. Vi ser då att uppgiften består av sex huvudsakliga delmoment:

1. Läsa in en given .csv-fil med data
2. Identifiera den information som ska sparas i en struct
3. Simulera MagicK1-strategin baserat på inläst data
4. Rita histogram över utfallet och linjegrafer över sista biten av data
5. Skriva formaterad data till Command Window och till fil
6. Skapa ett modulbaserat program där steg 1-2 utförs i en funktion, 3 i en annan och 4-5 i en tredje.

Ett förslag är att göra uppgiften i ordningen 1-2-3-4-5-6, d v s man kan börja skriva ett långt script och först på slutet dela upp det i tre funktioner, om man vill fokusera på själva huvudproblemet i början och inte fastna i funktionsdefinitionerna. **Testa programmet efter varje delsteg**. För att enklare kunna verifiera att mellanresultaten är riktiga, kan du t ex tillfälligt ta bort semikolonet på slutet av en rad.

Följande avsnitt i kursboken/Matlabhjälpen är extra viktiga för just denna inlämningsuppgift:

- Inläsning av data från .csv-fil (Kapitel 9.2-3)
- Hantering av strukturer och cellmatriser (Kapitel 8.1 & 8.2)
- Funktionsfiler (Kapitel 3.7 & 10.1)
- Skriva formaterad text till fil (Kapitel 3.3.2 & 9.3)
- `if`-satser, `for`-loopar, `while`-loopar (Kapitel 4.1-2 & 5.1-4)
- Histogram (Kapitel 12.1.2)

Det är **starkt rekommenderat** att ha gjort åtminstone några övningar ur boken och övrigt material på varje område innan du kastar dig över uppgiften!

Formalia

Du *ska* använda **kommentarsatser** när du programmerar! Det är mycket viktigt! **Ditt huvudprogram såväl som dina funktionsfiler ska innehålla programhuvud** med **ditt namn och personnummer**, samt **en kortare förklaring** av vad de gör. Programhuvudet i funktionerna ska innehålla information om vad de olika in- och utargumenten representerar. Koden skall i övrigt vara kommenterad på ett sådant sätt att det går att följa vad den gör.

Elektronisk inlämning

Inlämningsuppgiften ska rapporteras genom att de nödvändiga filerna lämnas in elektroniskt via Canvas **senast fredagen den 16:e oktober klockan 23:59**. **Följande fyra filer ska bifogas:** `MagicK1.m`, `readStockData.m`, `simulateStrategy.m`, och `printAndPlotResult.m`. m-filerna ska vara **väl kommenterade samt innehålla en beskrivning** av vad de gör.

Glöm inte att fylla i **namn** i programhuvud i alla filer ordentligt (så att vi är säkra på vem som gjort uppgiften)! Vid misstanke om plagiat skickas ärendet till disciplinnämnden.

Du behöver **inte** skriva någon rapport. Ladda upp filerna en och en utan att packa ihop dem till en ZIP. Ladda inte upp de utdelade csv-filerna (vi har dem redan). Vid rättning kommer en femte ”hemlig” datafil användas, så det finns ingen mening i att hårdkoda information som rör filnamnen.

Lycka till med uppgiften!