Regler

- Student får lämna salen tidigast en timme efter tentans start.
- Vid toalettbesök ska pauslista utanför salen fyllas i.
- All form av kontakt mellan studenter under tentans gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av tentavakt i samband med tentans start samt under tentans gång.
- Frågor om specifika uppgifter eller om tentan i stort ska ställas via tentasystemet.
- Systemfrågor kan ställas till assistent i sal.
- Ingen uppgift rättas efter tentatidens slut.
- Ingen kompletteringsmöjlighet ges de sista tio minuterna.
- Så länge en uppgift inte har betyget U kan den kompletteras till godkänt.
- Inga elektroniska hjälpmedel får medtas. Mobiltelefon ska vara avstängd och ligga i jacka eller väska.
- Inga ytterkläder eller väskor får förvaras vid skrivplatsen.

Antal uppgifter	5
Antal sidor (exklusive denna)	5
Hjälpmedel	En pythonbok (tex Learning Python 4 ed.) Ett A4-ark med egna anteckningar

VÄND!

Betygssättning

Tentan innehåller fem uppgifter. För godkänt betyg krävs minst två avklarade uppgifter. För gränser för högre betyg, se tabell 1.

Tid (timmar)	Antal uppgifter	Betyg
3 + B	3	5
4 + B	4	5
4 + B	3	4
5	2	3

Tabell 1: Betygssättning

Bonus från labserien

Varje avklarad deadline i labserien ger 5 minuter förlängning av samtliga gränser för högre betyg i tabell 1 (symboliseras av B i tabellen). Observera att maxtiden fortfarande är 5 timmar och att bonusen endast är giltig vid detta tillfälle.

Information

Inloggning

Logga in på ditt normala IDA-konto men välj session “exam system”.

Följ menyvalen så långt det går tills du ska mata in ett engångslösenord. Tag fram ditt LiU-kort och visa det för tentavakten för att få detta lösenord.

Tentan

Tentan kommer publiceras elektroniskt i form av ett pdf-dokument med namn `exam.pdf` i katalogen `given_files` när tentan startar. Observera att katalogens innehåll inte är läsbar innan tentans start.

Avslutning

Efter att du loggat ut som vanligt ska du vänta ett tag och sedan trycka på knappen “Avsluta tentamen” när det är möjligt. När detta är gjort är det omöjligt att logga in igen.

Uppgift 1

I programmet excell har man alltid en stor tabell med (i princip) obegränsat antal rader och kolumner. Raderna är numrerade från 1 och uppåt, men kolumnerna har lite speciell numrering. De börjar på 'A' och går sedan hela vägen till 'Z'. Efter detta så början man med "AA", "AB", "AC", o.s.v. Efter ett tag får man slut på sådana kombinationer också, då blir det: "AAA", "AAB", "AAC" o.s.v. Skriv ett program där användaren får mata in kolumnnummret och som sedan skriver ut motsvarande symbol som den hade sett ut i excell. Ingen felhantering krävs. Din lösning ska vara rekursiv.

Körexempel 1

```
Mata in ett kolumnnummer: 4
I Excell är detta kolumn D.
```

Körexempel 2

```
Mata in ett kolumnnummer: 26
I Excell är detta kolumn Z.
```

Körexempel 3

```
Mata in ett kolumnnummer: 27
I Excell är detta kolumn AA.
```

Körexempel 4

```
Mata in ett kolumnnummer: 28
I Excell är detta kolumn AB.
```

Körexempel 5

```
Mata in ett kolumnnummer: 1024
I Excell är detta kolumn AMJ.
```

Uppgift 2

En ”rak talsekvens” är en sekvens av heltal där skillnaden mellan två närliggande tal är samma oavsett vilka tal vi undersöker. Exempelvis är sekvenserna ”1 2 3 4” och ”4 7 10” raka medan ”3 5 7 8” inte rak.
Din uppgift är att skapa ett program som, givet ett filnamn på kommandoraden, läser genom en fil och skriver ut hur många sekvenser som fanns med i filen (en sekvens per rad) samt hur många av dem som INTE var raka.

Om ingen fil anges eller om filen inte kan läsas ska programmet avslutas med ett bra felmeddelande.

Du kan anta att varje sekvens har minst två tal och att de är ordnade i stigande ordning.

Körexempel 1

```
$ raka.py given_files/sekvenser.txt
Antal sekvenser: 5
3 var inte raka.
```

Körexempel 2

```
$ raka.py given_files/sekvenser2.txt
Antal sekvenser: 15
6 var inte raka.
```

Uppgift 3

I brädspelet *Catan* (tidigare Settlers of Catan) är en stor del av spelet resurshantering. Det finns flera resurser som kan användas för att exempelvis bygga vägar och städer. Resurser kan fås utanför spelarens tur, men under min egen tur kan jag även byteshandla resurser. Alla resurser kan bytas med banken i förhållandet 4:1 (dvs jag kan byta fyra resurser av samma typ mot en av någon annan), men jag kan också under spelet skaffa mig bättre förutsättningar.
Din uppgift är att skapa ett program som låter användaren mata in de resurser och bytesförhållanden som nu råder. Därefter matar användaren in vilken resurs hen behöver mer av (egentligen vore det praktiskt att säga exakt vilken kombination av resurser som krävs, men vi nöjer oss här med en resurs). De två inmatningarna separeras med en tomrad. Ditt program ska därefter antingen skriva ut vilka byten som måste göras eller skriva ut att det inte är möjligt.
Varje rad i inmatningen ser ut på följande format:

```
Resurs  Antal  [("ResursX" "Antal X som behövs för att få en Resurs")...]
```

Följande inmatning betyder då att spelaren har 3 av resursen ”Trä” och att det krävs 4 ”Sten” eller 2 ”Får” för att få en ”Trä”.

```
Trä 3 Sten 4 Får 2
```

Körexempel 1

```
Välkomen till Catan!
Mata in dina resurser:
Trä 3 Sten 2 Får 4 Metall 3
Sten 1 Trä 4 Får 4 Metall 4
Får 2 Trä 3 Sten 1 Metall 2
Metall 2 Sten 4 Får 2 Trä 3


Mata in önskad resurs:
Sten

Du kan byta.
2 Metall -> 2 Får
4 Får -> 1 Sten
```

Observera att även bytet ”2 Får -> 1 Metall, 3 Metall -> 1 Trä, 4 Trä -> 1 Sten” är giltig. Vilken du hittar spelar ingen roll.

Du kan anta att alla resurstyper matas in och att alla rader i inmatningen innehåller alla resurser.

Uppgift 4

I denna uppgift ska du skapa en funktion `is_ordered` som tar emot en samling data (exempelvis en lista) och returnerar `True` om värdena är ordnade. Normalt sett ska värden jämföras med vanliga `<` operatorn, men användaren ska även kunna ange en funktion som används för att jämföra två objekt. Denna funktion ger `True` om det första argumentet anses vara mindre än (ska komma före) det andra.

Filen `given_files/ordered.py` innehåller några testfall där du ser hur funktionen används. Kopiera denna fil och lägg till din definition här. Du får gärna lägga till egna testfall.

Uppgift 5

I denna uppgift ska du skapa en ADT för att hantera klockslag samt ett enkelt program som använder denna ADT. Följande funktioner ska finans i din ADT (på filen `time.py`), andra funktioner du behöver är också tillåtet att lägga till:

create_time Tar emot tre heltal (timme, minut och sekund) och ger tillbaka ett klockslag.

from_str Tar emot en sträng på formatet `HH:MM:SS` och returnerar ett klockslag.

add_seconds Tar emot ett klockslag och ett antal sekunder. Funktionen ger tillbaka ett nytt klockslag som är så många sekunder efter det angivna.

to_str Tar emot ett klockslag och ger tillbaka en sträng på formatet `HH:MM:SS`.

valid_time Tar emot ett klockslag och returnerar `True` om det är ett giltigt klockslag (timme i intervallet $[0, 23]$, minut och sekund i intervallet $[0, 59]$).

Skapa även ett program som använder modulen för att be användaren mata in ett klockslag. Så länge det inmatade är felaktigt ber programmet om ett nytt klockslag. När ett korrekt klockslag matats in ska nästa klockslag skrivas ut (dvs en sekund senare).

Körexempel 1

```
Mata in ett klockslag: 02:76:00
FELAKTIGT KLOCKSLAG
Mata in ett klockslag: 22:02:87
FELAKTIGT KLOCKSLAG
Mata in ett klockslag: 22:59:59
Nästa klockslag: 23:00:00
```