Regler

- Student får lämna salen tidigast en timme efter tentans start.
- Vid toalettbesök eller rökpaus 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 deadline du mött i labserien under kursens gång ger dig fem minuter extra tid för gränserna för högre betyg (symboliseras med B i tabell 1). Observera att bonus endast gäller detta tillfälle och inte ger förlängd sluttid på tentan.

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 spelet ”hänga gubbe” (eller hangman) utses en person till ritare. Ritaren ska hitta på ett ord som övriga deltagare ska försöka gissa sig fram till. Ritaren visar antal bokstäver i ordet genom att rita så många horisontella sträck på en bit papper eller en whiteboard. Därefter föreslår de andra en bokstav i taget. Om bokstaven finns med i ordet skriver ritaren ut bokstaven på rätt plats över sträcken. Om bokstaven inte fanns med ritar ritaren ut ett streck på en figur som till slut liknar en hängd sträckgubbe. Leken fortsätter tills en av deltagarna gissar rätt ord eller tills ritaren fått rita hela figuren.

I denna uppgift ska du skapa spelet hangman där spelet agerar ritare. Du behöver dock inte lyckas rita ut sträckgubben utan vi anger istället antal gissningar. I filen `given_files/words.txt` finns en lista över ord (ett ord per rad) varifrån programmet ska slumpa ett ord som användaren ska försöka lista ut.

Du kan utgå ifrån att orden i filen endast är med gemener (små bokstäver) och att användaren endast matar in gemener. Vi kan i detta program inte gissa på hela ordet utan användaren matar alltid in en bokstav åt gången.

Körexempel 1 (Användarinmating i ***kursiv stil***)

```
Välkommen till hangman.

Nuvarande ord: _ _ _ _ _
Gissade bokstäver:
Antal gissningar kvar: 11
Din gissning: e

Det var rätt!
Nuvarande ord: _ e _ _ _
Gissade bokstäver: e
Antal gissningar kvar: 11
Din gissning: i

Det var fel!
Nuvarande ord: _ e _ _ _
Gissade bokstäver: ei
Antal gissningar kvar: 10
Din gissning: i

Redan använt i.
Ny gissning: a

Det var rätt!
Nuvarande ord: _ e _ _ a
Gissade bokstäver: eia
Antal gissningar kvar: 10
Din gissning: w

Det var rätt!
Nuvarande ord: _ e _ _ a
Gissade bokstäver: eiaw
Antal gissningar kvar: 9
Din gissning: t

Det var rätt!
Nuvarande ord: t e _ t a
Gissade bokstäver: eiat
Antal gissningar kvar: 9
Din gissning: n

Det var rätt!
Grattis du vann!
```

Uppgift 2

På vanliga skriftliga tentor brukar stora rektangulära salar med bänkar ståendes i rutnät användas. Vid ett tentatillfälle har tentamensvakterna bestämt sig för följande:

1. I första hand ska en student sätta sig vid en bänk så långt till vänster som möjligt
2. Om det finns flera lediga platser längst till vänster ska studenten sätta sig så långt fram (uppåt i exemplen) som möjligt.

I denna uppgift ska du göra ett program som med en figur visar vilka bänkar som kommer användas med "`[]`" och vilka bänkar som inte ska användas med "`><`" enligt nedan.

Körexempel 1

```
Mata in antal rader: 3
Mata in antal kolumner: 5
Mata in antal tentander: 12

[] [] [] [] ><
[] [] [] [] ><
[] [] [] [] ><
```

Körexempel 2

```
Mata in antal rader: 8
Mata in antal kolumner: 10
Mata in antal tentander: 62

[] [] [] [] [] [] [] [] >< ><
[] [] [] [] [] [] [] [] >< ><
[] [] [] [] [] [] [] [] >< ><
[] [] [] [] [] [] [] [] >< ><
[] [] [] [] [] [] [] [] >< ><
[] [] [] [] [] [] [] [] >< ><
[] [] [] [] [] [] [] >< >< ><
[] [] [] [] [] [] [] >< >< ><
```

Körexempel 3

```
Mata in antal rader: 2
Mata in antal kolumner: 2
Mata in antal tentander: 3

[] []
[] ><
```

Körexempel 4

```
Mata in antal rader: 5
Mata in antal kolumner: 1
Mata in antal tentander: 4

[]
[]
[]
[]
><
```

Uppgift 3

Börja med att kopiera den givna filen `given_files/uppgift3.py` till din hemkatalog. Den innehåller ett kort testprogram för att testa din lösning. Du får inte göra några modifikationer i den givna koden, men får gärna lägga till testfall vid behov.

Reguljära uttryck används för att matcha ett uttryck mot en sträng. Formatet på uttrycket kan vara ganska avancerat, men i denna uppgift ska du göra en begränsad matchning. Du ska i denna uppgift skapa en funktion som avgör om en sträng matchar ett given mönster innehållandes bokstäver samt de speciella tecknen `.`, `?` samt `*` enligt nedanstående algoritm. Två tecken *matchar* varandra om de är samma tecken eller om mönstrets tecken är '`.`' (se punkt 2 i algoritmen).

1. Om strängen är tom: matchar om mönstret är tomt eller om mönstret kan ignoreras (har två tecken där det andra är '`?`' eller '`*`')
2. Annars om både strängen och mönstret är ett tecken långa: matchar om första tecknet *matchar* (samma tecken eller mönstret är '`.`')
3. Annars om mönstret endast är ett tecken långt: matchar inte
4. Annars om mönstrets andra tecken är '`?`': Matchar om
   (a) första tecknet *matchar* samt att resten av mönstret (tecken 3 till slutet) matchar resten av strängen (tecken 2 till slutet), eller
   (b) resten av mönstret (tecken 3 till slutet) matchar hela strängen
5. Annars om mönstrets andra tecken är '`*`': Matchar om
   (a) första tecknet *matchar* samt hela mönstret matchar resten av strängen (tecken 2 till slutet), eller
   (b) resten av mönstret (tecken 3 till sista) matchar hela strängen
6. Annars matchar mönstret om första tecknet *matchar* strängens första tecken samt resten av mönstret matchar resten av strängen.

Uppgift 4

I filerna `given_files/card.py` och `given_files/deck.py` finns det ADTer för att hantera ett kort och en kortlek. Din uppgift är att använda dessa för att göra ett program som beräknar vilka kort croupiern i kortspelet black jack ska ta. Croupiern ska ta kort så länge totalsumman av de dragna korten är under 17. I spelet Black Jack har alla kort (oavsett färg) med valör 2 till 10 samma värde som sin valör. Alla klädda kort (knekt, dam och kung) har värde 10 och ess är värda 1 eller 11. Ni kan dock i denna uppgift räkna ess som värde 1.

Er uppgift är alltså att skapa en kortlek, blanda denna och ta nya kort tills summan av kortens värde enligt ovan är 17 eller mer. Du ska sedan skriva ut de kort som drogs enligt nedanstående körexempel.

Körexempel 1

```
Croupiern drog följande kort:
hjärter 2, ruter 7, spader kung
Totalsumman blev 19
```

Uppgift 5

Skriv ett program som räknar ut det minsta N där antalet avslutande nollor i talet $N!$ är minst X stycken (där användaren matar in X i intervallet $[0, 5000]$). Det är som du säkert inser direkt i princip omöjligt att räkna ut $N!$ givet att det är stora N så det finns (såklart) en bättre väg att gå än att göra detta och sen i efterhand räkna ut antalet nollor. Hur gör man då detta? En tanke är att man hittar alla primtalsfaktorer i N! så att man får fram alla de minsta faktorerer som i slutänden behöver multipliceras ihop för att få fram resultatet. Dessa primtalsfaktorer kan sen "klumpas ihop" på så sätt att man får fram alla kombinationer av tal som kan ge en nolla i slutet av N! De tal man då behöver ta hänsyn till för att få nollor i slutet är talen 10, 100, 1000, ... (men dessa är ju inte primtal så de dyker inte upp bland faktorerna). Tänker man till lite till får man nog fram att de enda tal som är intressanta bland primtalsfaktorerna är talen 2 och 5 (då dessa ger 10 som resultat).
Frågan kvarstår då hur många $2 * 5$-par som dyker upp när man primtalsfaktoriserat N!. När man funderar ett tag till kommer man fram till att det kommer räcka att räkna ut hur många 5:or som dyker upp då det finns fler 2:or än 5:or. Antalet 5:or är detsamma som antalet nollor i slutet av N!.
Här kan man nu tänka på flera olika sätt. Du får lösa problemet på vilket sätt du vill, men jag ger ett tips extra för att man inte skall sitta fast. Om man har talet 15! så kommer detta att se ut som följer: $1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 * 11 * 12 * 13 * 14 * 15$ Av dessa tal kan bara 5, 10 och 15 ge primtalsfaktorn 5. Dessa ger dessutom en 5:a var och en. Man kommer att se att vart femte tal ger en 5:a. Det som ställer till det lite är att t.ex. talet 25 ger två 5:or när man primtalsfaktoriserar. Det gäller även talen 50, 75, 100, ... Detta ger alltså att vi måste räkna alla dessa som en extra 5:a. Du inser säkert att det finns fler "fallgropar" i detta. Talen $5^3 = 125$, $5^4 = 625$, $5^5 = 3125$ är också luriga (och även deras multipler). Det man kommer fram till är att man måste hålla reda på alla 5^x-faktorer. Dessa ger extra 5:or.
När man funderat klart finns det en skapligt enkel formel som kan användas. Denna bygger i princip på att man tar N och delar detta tal med de olika 5^x som finns. Därefter adderar man alla dessa kvoter och vips har man antalet 5:or som dyker upp, d.v.s. antalet nollor i $N!$ är beräknat.

Körexempel 1

```
Mata in X: 0
Det finns 0 stycken 0:or i slutet av 0!
```

Körexempel 2

```
Mata in X: 5
Det finns 6 stycken 0:or i slutet av 25!
```

Körexempel 3

```
Mata in X: 1234
Det finns 1235 stycken 0:or i slutet av 4950!
```