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	3	5
4	4	5
4	3	4
5	2	3

Tabell 1: Betygssättning

Bonus från labserien

Vid omtentamen ges ingen extra bonus från labserien.

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

Din uppgift är att skriva ett program som gör en simulering för att kontrollera hur stor sannolikhet det är att minst två personer i en grupp fyller år på samma dag på året (vi antar att det är 365 dagar per år, inga skottår alltså). Det är givet att användaren matar in hur många personer som ingår i gruppen.

Själva simuleringen går till på så sätt att man kör (i programmet) ett test 10.000 gånger. Själva testet går till så att man slumpar ut vilken födelsedag de olika personerna har på året. Därefter kontrollerar man om det finns minst två som har samma födelsedag. Man räknar samtidigt hur många test som utfaller med att att det är minst två som fyller på samma dag. Till sist delar man antalet med 10.000 (förstås) och får då fram sannolikheten.

Om du vill lösa detta i annan ordning är det helt ok, men det är alltså inte någon matematisk formel man skall komma fram till utan det kommer att blir ett närmevärde på hur stor sannolikheten är. Resultatet skulle kunna bli helt galet om man har riktigt otur (då slumpen ibland spelar oss ett spratt), men att ha otur 10.000 gånger låter ju lite tungt.

Körexempel 1 (Användarinmatning i fet stil)

```
Mata in antalet personer: 2
Sannolikheten att minst två av dessa
fyller år på samma dag är ungefär: 0.24%
```

Körexempel 2

```
Mata in antalet personer: 5
Sannolikheten att minst två av dessa
fyller år på samma dag är ungefär: 2.86%
```

Körexempel 3

```
Mata in antalet personer: 23
Sannolikheten att minst två av dessa
fyller år på samma dag är ungefär: 50.38%
```

Körexempel 4

```
Mata in antalet personer: 32
Sannolikheten att minst två av dessa
fyller år på samma dag är ungefär: 75.68%
```

OBS! Dessa resultat är de vi fick vid en testkörning per gruppstorlek. Det kan alltså bli lite annorlunda resultat för er, men det rör sig i dessa trakter. Vid 7 ytterligare körningar med indata 5 fick jag följande resultat (2.60%, 2.60%, 2.55%, 2.43%, 2.40%, 2.58%, 2.98%).

Introduktion till uppgift 2-3

Följande två uppgifter baseras på labyrintlösning. I uppgift 2 ska du lösa en labyrint enligt en given algoritm och generera en lista av vägar som tas för att hitta den kortaste lösningen. I uppgift 3 ska du givet en labyrint och en lösning rita ut lösningen grafiskt med hjälp av några givna funktioner.

En labyrint representeras här med en lista av listor där varje element kan ha värden enligt tabell 2. Varje rad i labyrinten motsvarar en rad i den yttre listan och alla rader är lika långa.

En *lösning* på en labyrint är en lista med riktningar som måste tas för att ta sig från startrutan, som alltid är övre vänstra hörnet (koordinat (0,0)), till målet. Möjliga värden i listan ges av tabell 3 och exempel på två lösningar på labyrinten i figur 1 ges i kodexempel 1. Det är givet att en labyrint alltid har exakt en målpunkt.

Värde	Betydelse
0	Tom ruta
1	Vägg
2	Mål

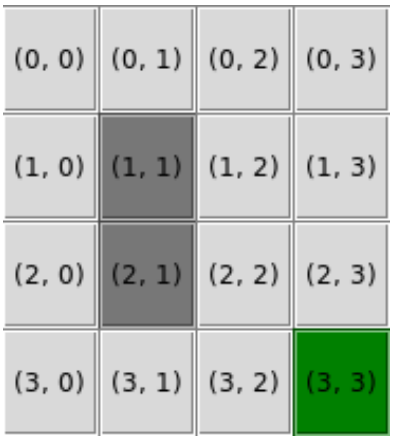
Tabell 2: Möjliga värden i en labyrint

Värde	Riktning
u	Uppåt
d	Nedåt
l	Vänster
r	Höger

Tabell 3: Möjliga riktningar i en lösning

Figuren nedan visar på samma labyrint representerad som en lista samt grafiskt. Observera att koordinaterna även är utskrivna på formatet (rad, kolumn).

```
[
    [ 0, 0, 0, 0 ],
    [ 0, 1, 0, 0 ],
    [ 0, 1, 0, 0 ],
    [ 0, 0, 0, 2 ]
]
```



Figur 1: Samma labyrint representerad både i kod och grafiskt.

```
sol1 = ['d', 'd', 'd', 'r', 'r', 'r']
sol2 = ['r', 'r', 'd', 'r', 'd', 'd']
```

Kodexempel 1: Två lösningar till labyrinten i figur 1

Uppgift 2

Denna uppgift går ut på att lösa en labyrint enligt en given algoritm. Du ska skapa en funktion som tar emot en labyrint och en koordinat (två heltal) och därefter returnerar den kortaste lösningen på labyrinten (minst antal steg som måste tas från koordinaten till målplatsen). Din funktion ska implementera nedanstående rekursiva algoritm.

1. Skapa en djup kopia av den givna labyrinten. Detta går att göra manuellt eller med funktionen `deepcopy` från modulen `copy`.
2. Markera nuvarande koordinat som besökt genom att sätta ett värde för koordinaten i kopian som skapades ovan. Exempelvis kan värdet -1 eller 3 användas för att markera en besökt punkt.
3. För varje närliggande koordinat
   (a) Om punkten innehåller målet: returnera dess riktning
   (b) Om punkten är giltig, gör ett rekursivt anrop för att kontrollera om det finns en giltig lösning i den punkten. En punkt är giltig omm (om och endast om):
      i. Koordinaten är giltig (varken rad eller kolumn är icke-negativ eller längre än längden på labyrinten i respektive dimension).
      ii. Värdet på positionen är inte en vägg eller redan besökt.
4. Om det efter steg 3.b finns flera lösningar returneras riktningen åt det kortaste hållet.
5. Om flera riktningar är kortast och lika långa ska labyrinten traverseras i följande ordning:
   (a) Nedåt
   (b) Höger
   (c) Vänster
   (d) Uppåt
6. Om ingen riktning är giltig i nuvarande koordinat returneras värdet `None`.

Följer man denna algoritm blir lösningen för labyrinten i inledningen på föregående sida enligt `sol1` i kodexempel 1.

OBS: I denna uppgift är det givet att labyrinten som skickas in har minst en giltig lösning.

Slutligen ska du skapa ett program som skapar en tom labyrint med storleken 5x5 som har målet i koordinat (4,4), dvs nedre högra hörnet. Lös denna väldigt enkla labyrint med din funktion - svaret bör vara `['d', 'd', 'd', 'd', 'r', 'r', 'r', 'r']`.

Uppgift 3

I denna uppgift ska du utifrån en given labyrint och lösning rita ut vägen som tas från start till mål. För att lyckas med detta finns det i modulen `given_files.draw` följande funktioner:

draw_grid(rows, cols) Ritar ut ett rutnät med storlek (rows, cols) i ett fönster på skärmen. Returnerar ett `handtag` till detta fönster.

wall(h, row, col) Markerar koordinat (row, col) som en vägg i rutnätet som handtaget h anger.

path(h, row, col) Markerar (row, col) som ett steg i lösningen.

goal(h, row, col) Markerar (row, col) som målplats.

pause(t) Ger en paus i programmet om `t` sekunder.

I filen `given_files/labyrinth.py` finns det given kod som testar att rita ut ett antal labyrinter och lösningar. Din uppgift är att implementera den tomma funktionen `draw_solution`. Kopiera filen till din arbetskatalog och modifiera denna. Du får inte göra ändringar i den givna koden utöver den tomma funktionen.

Tips: För att se att det blir korrekt rekommenderas det starkt att använda funktionen `pause` mellan varje steg i lösningen.

Uppgift 4

Citatet ”Det finns bara 10 sorters människor. De som förstår binära talsystemet och de som inte gör det.” är en gammal goding som ni kanske hört talas om.

I denna uppgift gäller det att skriva ett program som tar emot 3 positiva heltal som indata (direkt från kommandoraden) och skriver ut det heltal man får om man tar ”det första talet” tolkat som ett tal i talbasen ”det andra talet” och omvandlar detta till talbasen ”det tredje talet”.

Antag att man startar programmet på följande sätt:

```
> convert_base.py 13 10 2
```

Detta skulle innebära att vi skall omvandla talet 13 i basen 10 (vanliga decimala talsystemet) till basen 2 (binära talsystemet). Programmet skulle i detta fall ge som utskrift:

```
Talet 13 (bas 10) = 1101 (bas 2).
```

Fullständig rimlighetskontroll ska göras på argumenten, både på antal och att de har korrekta värden i sina respektive talbaser. Programmet ska endast fungera för talbaser i intervallet $[2, 16]$.

Körexempel 1

```
> convert_base.py 42 10 2
Talet 42 (bas 10) = 101010 (bas 2).
```

Körexempel 2

```
> convert_base.py 42 10 7
Talet 42 (bas 10) = 60 (bas 7).
```

Körexempel 3

```
> convert_base.py 4A 10 8
4A är inte giltigt i bas 10.
```

Körexempel 4

```
> convert_base.py 101010 C 10
Talbas C felaktig.
```

Körexempel 5

```
> convert_base.py 1010 8 15
Talet 1010 (bas 8) = 24a (bas 15).
```

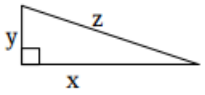
Uppgift 5

Ett populärt AR-spel (Augmented Reality) är *Ingress*. I spelet är man på två lag som försöker ta över varandras portaler. Portalerna i spelet motsvarar ofta något landmärke eller sevärdhet i verkligheten. För att ta över en portal måste man (bl.a.) vara fysiskt i närheten av den. Spelet är alltså utmärkt för den som är ute och rör på sig. En flitig spelare kommer alltså ta sig runt rätt mycket och det kan bli en hel del kilometrar varje gång man spelar spelet!

På filen `given_files/INGRESS.TXT` har vi ett antal portaler som en spelare vill besöka. Portalerna representeras av två tal som är den position (en X- och Y-koordinat) som portalen finns på (enheten är kilometer). Portalerna i filen förekommer i den ordning som man tänker besöka dem. Givet att man alltid går rakt från en portal till en annan kan man då fråga sig hur långt det blir om man skall besöka alla. Om vi exempelvis studerar följande punkter:

```
3 3
1 6
1 8
```

Avståendet mellan den första och den andra portalen är $\sqrt{(1-(-3))^2 + (6-3)^2} = 5$ enligt pythagoras sats (figur 2)
Avståndet mellan den andra och den tredje portalen är $\sqrt{(1-1)^2 + (8-6)^2} = 2$
Det totala avståendet blir alltså 7.0 km.



Figur 2: Pythagoras sats, $x^2 + y^2 = z^2$

Avstånden i filen är alltid heltal, men resultatet är ett reellt tal. Du hittar funktionen för roten ur (`sqrt`) i modulen `math`.

Skriv ett program som läser igenom filen och beräknar det totala avståendet mellan portalerna på ovanstående vis. Du kan utgå ifrån att filen har minst två punkter (d.v.s. minst två rader).

Körexempel 1

```
Det totala avståendet mellan portalerna i filen är 56.3 km.
```