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 (inklusive denna)	8
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

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

Kirk har slut på pengar denna månad och funderar på att ta ett lån för att slippa leva på nudlar tills CSN kommer in igen. Att beräkna hur lång tid det tar att betala tillbaka ett lån kan vara lite krångligt och Kirk tycker nog att 50% ändå inte låter som så mycket ränta varje månad. Men han är trots allt vän med en programmerare och bestämmer sig för beställa ett litet program som kan berätta för honom hur snabbt han kan betala tillbaka lånet och hur mycket ränta han får betala i slutänden.

Han bestämmer sig för att programmet ska fungera enligt körexemplet nedan. Du kan räkna med att användaren är kompetent och gör inmatning enligt exemplet nedan. Observera att summan av räntan skall rundas av till 1 decimal. Räkna med att varje månads ränta läggs till på den lånade summan innan Kirk kommer hinna betala av det han kan den månaden.

Körexempel 1 (Användarinmatning i fet stil)

```
> python3 Uppgift1.py
Hur mycket ränta vill "banken" ha?
40
Hur mycket vill du låna och hur mycket kan du betala tillbaka varje månad?
Jag vill låna 1000kr och kan betala tillbaka 500kr i månaden
Efter 5 månader har du betalat tillbaka de 1000 kronorna samt 1405.4 kr ränta
```

Körexempel 2 (Användarinmatning i fet stil)

```
> python3 Uppgift1.py
Hur mycket ränta vill "banken" ha?
40
Hur mycket vill du låna och hur mycket kan du betala tillbaka varje månad?
Jag vill låna 2000kr och kan betala tillbaka 500kr i månaden
Du kommer aldrig kunna betala tillbaka detta lån
```

Körexempel 3 (Användarinmatning i fet stil)

```
> python3 Uppgift1.py
Hur mycket ränta vill "banken" ha?
10
Hur mycket vill du låna och hur mycket kan du betala tillbaka varje månad?
Jag vill låna 100kr och kan betala tillbaka 11kr i månaden
Efter 26 månader har du betalat tillbaka de 100 kronorna samt 176.8 kr ränta
```

Uppgift 2

Picard gillar att spela figurspel. I figurspelet Warhammer 40k används tärningar för att bestämma om man lyckas med något eller inte. Tärningar anges vanligtvis i formatet IDX där I är antalet tärningar och X är antalet sidor på varje tärning. 1D6 är alltså en tärning med sex sidor. 3D10 är tre stycken tärningar med tio sidor vardera.

Det är vanligt att man ska rulla högre eller lägre än ett visst värde med en eller fler tärningar i fugurspel för att bestämma om man lyckas med att exempelvis storma en annan enhet. Det är i sådana fall bra att veta hur stor chansen är att man lyckas i förväg. Picard är inte nödvändigtvis korkad och kan utan större problem förstå hur stor chansen är att slå högre än 3 med 1D6 men när det är flera tärningar inblandade får han lätt huvudvärk när han försöker räkna.

Han har därför beställt ett litet program från dig som skall hjälpa honom förstå fördelningen av resultaten när man slår flera tärningar. Programmet ska generera alla kombinationer tärningarna kan skapa (36 stycken för 2D6, 9 stycken för 2D3). Detta är alla nio kombinationer för 2D3:

1-1	1-2	1-3
2-1	2-2	2-3
3-1	3-2	3-3

Varje kombination ska sedan summeras för att få fram det motsvarande resultatet (kombinationen 1-3, 3-1 och 2-2 ger till exempel resultatet 4). I en lämplig datatyp ska du sedan spara hur många gånger varje resultat förekommer. För 2D3 ser fördelningen av resultat ut som följer:

Resultat:	Antal förekomster:
2	***1***
3	***2***
4	***3***
5	***2***
6	***1***

När du sparat fördelningen av resultaten på det sättet ska de bara skrivas ut i två kolumner där den första kolumnen är resulatet och den andra kolumnen innehåller en stjärna för varje kombination av tärningar som kan leda till det resultatet. Programmet ska fungera för godtyckligt antal tärningar med godtyckligt antal sidor. Programmet ska fungera som körexemplen på nästa sida:

Körexempel 1 (Användarinmatning i fet stil)

```
> python3 Uppgift2.py
Mata in antalet tärningar:
2
Mata in antalet sidor för tärningarna:
3
Resultat:
2   *
3   **
4   ***
5   **
6   *
```

Körexempel 2 (Användarinmatning i fet stil)

```
> python3 Uppgift2.py
Mata in antalet tärningar:
3
Mata in antalet sidor för tärningarna:
6
Resultat:
3   *
4   ***
5   ******
6   **********
7   ***************
8   ********************
9   *************************
10  *****************************
11  *****************************
12  *************************
13  ********************
14  ***************
15  **********
16  ******
17  ***
18  *
```

Uppgift 3

Sisko är inte nödvändigtvis den skarpaste kniven i lådan och hans son, Jake, envisas med att prata på krypterat språk med sin vän. Framförallt använder han rövarspråket som uppfanns av Astrid Lindgren för Kalle Blomkvist där varje konsonant dubblas och ett o sätts in mellan dem. Vokaler och andra tecken lämnas oförändrade. Exempelvis blir det svenska ordet "apa" apopa på rövarspråket.

Din uppgift här är att skapa ett program åt Sisko så han kan uppfylla sin plikt som förälder och övervaka allting sonen säger. Han ska kunna mata in en fras (en rad) på rövarspråket och sedan ska programmet skriva ut frasen på svenska.

Körexempel 1 (Användarinmatning i fet stil)

```
> python3 Uppgift3.py
Mata in en fras på rövarspråket: hohejoj jojagog hohetoteror Anondoderorsos!
Frasen på svenska: hej jag heter Anders!
```

Körexempel 2 (Användarinmatning i fet stil)

```
> python3 Uppgift3.py
Mata in en fras på rövarspråket: Anonnona kokokokaror kokafoffofe
Frasen på svenska: Anna kokar kaffe
```

Uppgift 4

Alla besättningsmedlemmar som tjänstgör under kapten Janeway har ett särskillt nummer som identifierar dem. På grund av lite strul med datorerna ombord har den sista siffran i dessa nummer försvunnit. Hon vänder sig därför till dig för att fixa den trasiga datan.

Som tur är den sista siffran i detta identifieringsnummer en kontrollsiffra som genereras enligt mod-10-algoritmen. Enlig denna algoritm multipliceras siffrorna växelvis med 2 och 1 och därefter summeras produkterna. För en tvåsiffrig produkt beräknas summan av dess siffror. Summan dras sedan bort från närmsta övre tiotal för att få fram kontrollsiffran.

För att beräkna kontrollsiffran för identifikationsnummret `8112189870` utförs följande beräkning:

```
   8 1 1 2 1 8 9 8 7 0
*  2 1 2 1 2 1 2 1 2 1
------------------------
   ^ ^ ^ ^ ^ ^ ^ ^ ^  ^
  16 1 2 2 2 8 18 8 14 0
```

$$1 + 6 + 1 + 2 + 2 + 2 + 8 + 1 + 8 + 8 + 1 + 4 + 0 = 44$$

Kontrollsiffran är $50 - 44 = 6$

I filen ident_numbers.json hittar du en lista över alla besättningsmedlemmars första 10 siffror. Skriv ett program som läser in alla dessa siffror och skriver ut dem med den beräknade kontrollsiffran. Programmet ska fungera enligt körexemplet nedan och du behöver inte felkontrollera filnamn etc.:

Körexempel 1

```
> python3 Uppgift4.py ident_numbers.json
81121898706
91121898704
71121898708
18688997999
65379139264
88293231427
21279158121
51598354290
63549678573
62439475430
56459458730
69224988153
84558346312
```

Uppgift 5

Archer har glömt koden till sin dörr. Han tycker nog att det inte finns så många olika kombinationer som det kan vara eftersom han kommer ihåg alla inblandade siffror, han vet bara inte i vilken ordning de skall skrivas in. Koden till dörren är 4 siffror lång och varje siffra är i intervallet 0-9.

Du har som uppgift att skriva ett program som frågar efter 1 till 4 siffror och sedan skriver ut antalet möjliga kombinationer dessa kan finnas med i. Om färre än 4 siffror fylls i antas vilka siffror som helst vara möjliga. Vid inmatningen 1, 2, 2 är exempelvis 2129 en möjlig kombination. Tänk på att om användaren matar in 2 av samma siffra måste en möjlig kod också innehålla två av den siffran. Om användaren matar in 1, 2, 2, 3 är 2312 en möjlig kod men inte 1233.

Programmet ska fungera enligt körexemplet nedan. Du behöver rimlighetskontrollera inmatningen:

Körexempel 1

```
> python3 Uppgift5.py
Mata in de siffror som ingår i koden: 1 2 3 4
Det finns 24 möjliga kombinationer.
```

Körexempel 2

```
> python3 Uppgift5.py
Mata in de siffror som ingår i koden: 4
Det finns 3439 möjliga kombinationer.
```

Körexempel 3

```
> python3 Uppgift5.py
Mata in de siffror som ingår i koden: 1 1 1
Det finns 37 möjliga kombinationer.
```

Körexempel 4

```
> python3 Uppgift5.py
Mata in de siffror som ingår i koden: 9 9 10 9
Felaktig inmatning. Mata in ett till fyra heltal i intervallet 1-9.
```