Teori: Variabler

En variabel har fyra grundläggande egenskaper: *Namn, Typ, Värde* och *Synlighet*.

NAMNET måste följa den namngivningsstandard som finns. Vanligtvis måste ett variabelnamn inledas med en bokstav, men kan innehålla siffror och en del andra tecken. Mellanslag och liknande är normalt inte tillåtet i variabelnamn. I Python går det att använda svenska tecken i variabelnamn, men det rekommenderas inte eftersom det kan bli problem i ett senare skede.

Några bra tumregler för variabelnamn som fungerar i de flesta sammanhang:

- Använd representativa variabelnamn
- Undvik att ha variabler som bara skiljer sig åt med stor/liten bokstav.
- Använd konsekvent enbokstavsvariabler såsom exempelvis **x**, **y**, **z**, **i**, **j**, **k** i slingor.
- Använd helst relativt korta variabelnamn för ökad läsbarhet

Varje variabel har en **TYP** som beskriver vilken typ av data som kan lagras i variabeln. Detta är nödvändigt eftersom all data i minnet representeras i binär form. Några exempel på vanligt förekommande typer är heltal (integer), decimaltal (float), sanningsvärden (boolean) samt strängar (string). En strängvariabel är en variabel som innehåller text bestående av ett eller flera tecken.

En variabels **VÄRDE** är helt enkelt det data som ligger lagrad i variabeln. Värdet måste matcha variabelns typ.

Den sista egenskapen hos en variabel är den **SYNLIGHET** som en variabel har. Detta handlar om i vilka delar av ett program som en variabel kan användas.

```
x = 5

def f(x) :
    return x * 2

print(x)            # Skriver ut talet 5
x = f(x)
print(x)            # Skriver ut talet 10
```

Om du till exempel skapar en funktion **`f(x)`** kommer värdet på **x** bara att vara synligt inom den funktionen. När funktionen avslutas glömmer datorn bort vad **x** var och du kan inte hänvisa till det utanför funktionen. Däremot kan du utanför funktionen ha ett *annat* **x** som har ett annat värde.

Det innebär att det är möjligt att skapa flera funktioner som också tar argument som heter **x** eftersom synligheten för varje **x** är begränsad till respektive funktion.

Det går att specificera att en variabel är *global* om man vill kunna manipulera den inne i olika funktioner. Mer om det senare.

Teori: Villkorsuttryck i slingor och alternativsatser

Ett *villkorsuttryck* är en utsaga som kan vara sann (**`True`**) eller falsk (**`False`**). Här följer tre exempel på villkorsuttryck:

```
3 < 2          x >= 5          x + 3 > 7 and y < 2
```

Det första villkorsuttrycket är alltid falskt eftersom 3 inte är mindre än 2. Värdet på de övriga två beror på vilka värden variablerna *x* och *y* har. I det sista villkorsuttrycket binder **+** hårdare än **>**, d.v.s. man jämför **`x + 3`** med **7**, inte bara 3:an. Detta är också ett sammansatt uttryck som genom operatorn **`and`** kräver att två olika villkor ska vara sanna för att hela uttrycket ska vara sant.

Följande operatorer kan användas i Python för att formulera villkorsuttryck:

```
<    >    <=   >=   ==   !=   is   not  and  or
```

Villkorsuttryck används framför allt till två uppgifter i ett dataprogram:

1. För att styra ett alternativ. Detta brukar även kallas för *if-satser*.
2. För att styra en slinga som är beroende av ett villkorsuttryck är sant

Betrakta följande kod:

```
i = 1
while i < 10 :
     if i == 5 :
          print(”i är nu fem!”)
     i = i + 1
```

`while`-slingan styrs av ett villkorsuttryck som gör att slingan upprepas ett varv till så länge *i* är mindre än 10. Inne i **`while`**-slingan finns en **`if`**-sats som kontrollerar om variabeln *i* har värdet 5; i så fall skrivs en särskild text ut.

Man kan också använda en **`while`**-slinga för att försäkra sig om att användaren av programmet anger ett giltigt svar (funktionen **`int`** omvandlar svaret till ett heltal (**int**eger):

```
svar = int(input("Ange ett tal större än 0:"))
while svar < = 0  :
     svar = int(input("Försök igen:"))
```

När man använder sig av **`if`**-satser så går det också att ha flera olika villkor, som då kommer att testas i den ordning de är angivna. I exemplet nedan är **`elif`** kort för "else if"

```
if i == 5 :
     print("i är nu fem!")
elif i == 7 :
     print("i är nu sju!")
else
     print("i är nu", i)
```

Övning 1

Betrakta funktionen **`isodd()`** nedan. Den inbyggda funktionen **`int()`** omvandlar dess argument (i detta fall ett decimaltal) till ett heltal (integer). **Skriv in funktionen i din editor.**

```
def isodd(x) :
    kvot = x / 2
    return ( kvot != int(kvot) )
```

a) Vad betyder **`!=`** i koden ovan?

b) Vilken *typ* har det svar som funktionen returnerar?

Den inbyggda funktionen **`input()`** kan användas för att läsa in ett svar från den som kör programmet och lagra den i en strängvariabel (d.v.s. som text).
Exempelvis så här:

```
x = input()
```

eller så här om du vill ange en ledtext före inmatningen:

```
x = input("Ange x :")
```

c) Skriv ett program som frågar efter ett tal, använder funktionen **`isodd()`** för att avgöra om talet är udda eller jämnt och sedan skriver ut antingen ”udda” eller ”jämnt”. Tänk på att **`input()`** lagrar sitt svar i en textsträng! **Tips**: utskrift görs med **`print()`**

Funktionen **`isodd()`** avgör om ett tal är udda eller jämnt genom att dela talet med två och se om decimaldelen är 0 eller inte. En annan variant är att använda *modulooperatorn* som i Python anges med procenttecknet **`%`** . D.v.s. om du skriver **`a % b`** så beräknas *a* modulo *b*.

d) Skriv om funktionen **`isodd()`** så att den använder modulooperatorn istället.

Collatz förmodan kan formuleras enligt följande:

1. Utgå från ett positivt heltal *n*.
2. Om *n* är jämnt, dividera det med två.
   Om det är udda, multiplicera det med tre och addera därefter ett.
3. Upprepa steg 2 tills du når talet ett.

Problemet i Collatz förmodan är om det går att nå talet ett för alla värden på *n* ?

e) Skriv en funktion **`collatz()`** som tar ett argument *n* och returnerar antingen $n/2$ eller $3*n + 1$ beroende på om *n* var udda eller jämnt. Funktionen skall använda sig av funktionen **`isodd()`** för att avgöra om talet är udda eller jämnt.

f) Skriv ett program som frågar efter ett tal *n* och därefter anropar funktionen **`collatz()`** upprepade gånger för att uppdatera värdet på *n* till dess att det blir 1. För varje anrop av **`collatz()`** skall det nya värdet på *n* skrivas ut.
Tips: här behöver du formulera ett villkorsuttryck för en slinga.

Teori: Inbyggda funktioner och användning av moduler

I föregående övning fick du bekanta dig med några av de funktioner som finns inbyggda i Python: **`input()`**, **`print()`** samt **`int()`**. Dessa funktioner sägs ingå i Pythons *standardbibliotek*.

En viktig förmåga vid programmering är att kunna sätta sig in i olika bibliotek, deras funktioner och hur man använder dem. I praktiken innebär det att kunna hitta i, och använda, den online-dokumentation som finns.

Standardbibliotekets dokumentation finns här: https://docs.python.org/3/library/ .

Betrakta följande funktion:

```
def intislen(n, le) :
    intlen = len(str(n))
    return (intlen == le)
```

Här ser du att en funktion kan ta fler än ett argument. Dessutom används två inbyggda funktioner som vi inte har presenterat tidigare:

`len()`	Ger längden (om den existerar) av sitt argument
`str()`	Omvandlar ett objekt till en textsträng (motsvararande **`int()`** för heltal)

Eftersom ett heltal inte har en längd som kan mätas med **`len()`** görs heltalet om till en sträng som sedan kan användas som argument till funktionen **`len()`**

Utöver de redan introducerade inbyggda funktionerna **`input()`**, **`print()`**, **`int()`**, **`len()`** och **`str()`** så är det bra att känna till nedanstående inbyggda funktioner. För närmare information om exakt hur de fungerar hänvisas till webbsidan för standardbibliotekets dokumentation, sektion 2.

`abs(x)`	Absolutbeloppet av x
`ascii(x)`	ASCII-värdet för argumentet som skall vara ett tecken
`chr(x)`	Ger tecknet med ASCII-värdet x (motsatsen till ascii())
`float(x)`	Översätt x till ett flyttal
`help()`	Inbyggd hjälp-funktion i Python
`list()`	Skapa en lista
`max(x)`	Hitta största värdet i x
`min(x)`	Hitta minsta värdet i x
`quit()`	Avsluta Python
`range(x)`	Ange ett intervall från 0 till x-1
`round(x)`	Avrunda x
`set()`	Skapa en mängd

Teori: Olika typer av slingor

Det finns två grundläggande typer av slingor som är vanligt förekommande i de flesta programspråk, så även i Python:

- **`for`**-slingan som upprepas ett förutbestämt antal gånger
- **`while`**-slingan som upprepar en del av programmet så länge ett angivet villkorsuttryck är uppfyllt

Vilken slinga man använder sig av beror alltså på förutsättningarna. **`for`**-slingan har till skillnad från **`while`**-slingan en *indexvariabel* som normalt ökar med 1 för varje varv i slingan. Följande exempel på **`for`**-slinga skriver ut alla heltal från 1 till 10:

```
for x in range(1, 11) :
     print(x)
```

Notera att **`range(a, b)`** går från *a* till *b-1*, därför behöver man ange slutvärdet + 1.

Det går också att använda **`range()`** med enbart ett argument; då kommer den att gå från 0 till det angivna argumentet-1. Följande **`for`**-slinga skriver också ut alla heltal mellan 1 till 10:

```
for x in range(10) :
     print(x + 1)
```

`while`-slingan har redan introducerats tillsammans med villkorsuttryck. I en **`while`** -slinga anges ett villkorsuttryck som styr om slingan skall köra ett varv till eller inte. För att en **`while`** -slinga skall fungera som tänkt är det mycket viktigt att det sker något inne i slingan som har möjlighet att påverka villkorsuttrycket så att det så småningom blir falskt och slingan avslutas. I annat fall kommer programmet att hamna i en "evighetsslinga".

Ett exempel på en funktion som använder en **`while`** -slinga för att beräkna största gemensamma nämnare för *a* och *b* med Euklides algoritm:

```
def gcd(a, b) :
    while b != 0 :
        t = b
        b = a % b
        a = t
    return a
```

Här vet vi inte på förhand hur många varv slingan skall utföras; bara att den skall utföras så länge **`b`** inte är 0.

Det går att utforma en **`while`**-slinga så att den fungerar som en **`for`**-slinga, men det anses vara dålig programmeringsstil. Genom att använda en **`for`** -slinga så signalerar du läsaren av koden att det är något som skall utföras ett förutbestämt antal gånger, vilket du inte gör lika tydligt om du använder en **`while`** -slinga där det borde ha varit en **`for`** -slinga.

Övning 2

En trevlig operator i Python är **`in`** som kan användas bland annat till att kontrollera om en sträng förekommer i en annan sträng. Exempelvis:

`"hej" in "hejsan"` har värdet **`True`**
`"han" in "hejsan"` har värdet **`False`**

Du ska nu lösa denna uppgift:
Hur stor andel av alla 7-siffriga tal utan inledande nollor innehåller minst en 3:a ?

a) Komplettera nedanstående skelettkod till en funktion så att funktionen löser uppgiften.

```
def find3s() :
    antal = 0
    for x in range(1000000,10000000) :
        if (...) :
            antal = antal + 1
    return ...
```

I koden ovan går **`for`** -slingan från 1000000 - vilket är det lägsta sjusiffriga talet utan inledande nollor - till 10000000 som är ett åttasiffrigt tal. Anledningen till detta är att när du skriver **`range(a, b)`** så kommer det att innebära alla tal från *a* till *b-1*. Vi behöver alltså ange det högsta sjusiffriga talet + 1 som slut för att testa mot alla sjusiffriga tal.

Du har kanske noterat att in-operatorn förekommer när man skapar en **`for`** -slinga? Anledningen till detta är att funktionen **`range(a,b)`** skapar en *lista* med *element* som går från *a* till *b-1.* Om du exempelvis skriver **`range(0, 5)`** skapas en lista med följande utseende:

```
[0, 1, 2, 3, 4]
```

`for`-slingan kommer att gå igenom de värden som finns i listan och spara dem i variabeln *x*. När hela listan har gåtts igenom och x har antagit alla värden som fanns i listan är for-slingan klar. Även om vi hittills har använt **`for`**-slingan till att gå från ett heltal till ett annat så kan den i praktiken användas för att gå igenom vilken lista som helst i tur och ordning.

Exempelvis kommer följande kod:

```
dagar = ['måndag', 'tisdag', 'torsdag']
for x in dagar :
    print(x)
```

att skriva ut denna text:

```
måndag
tisdag
torsdag
```

b) Modifiera funktionen **`find3s()`** så att den tar ett argument *n* och returnerar andelen av alla *n*-siffriga tal som har minst en 3:a.
Tips: a^b uttrycks i Python som **`a**b`**

Teori: Listor

En *lista* är en indexerbar variabel bestående av flera *element*. I andra programspråk och sammanhang kan samma konstruktion kallas för exempelvis *array*, eller *vektor*. Index i en lista börjar normalt på 0, vilket innebär att i en lista med 10 element är indexen till de olika elementen 0, 1, ... 9.

För listor finns det inbyggda funktioner som kan användas för att hantera dem. Om vi har en lista som heter **`Tal`** så kan man använda bland annat följande funktioner på den listan:

`len(Tal)`	antalet element i listan **`Tal`**
`min(Tal)`	minsta elementet i listan **`Tal`**
`max(Tal)`	största elementet i listan **`Tal`**
`Tal.count(x)`	förekomsten av elementet **`x`** i listan **`Tal`**
`Tal.append(x)`	lägg till elementet **`x`** i listan **`Tal`**
`Tal.sort()`	sorterar listan **`Tal`**

Notera att man för de tre sistnämnda funktionerna ovan måste ange listans namn innan funktionsnamnet.

Det finns några olika alternativ för att skapa en lista. Föjande två kodexempel skapar på två olika sätt en lista med namnet **`Tal`** som innehåller talen 1, 7 och 11:

```
Tal = [1, 7, 11]
```

```
Tal = []
Tal.append(1)
Tal.append(7)
Tal.append(11)
```

Om man vill skapa en lista som innehåller heltalen 1 till 10 kan man använda en **`for`**-slinga:

```
Tal = []
for x in range(1,11) :
     Tal.append(x)
```

Vi har redan nämnt att funktionen **`range()`** skapar listor. Det är inte helt sant, men man skulle kunna använda funktionen **`list()`** för att omvandla resultatet från **`range()`** till en lista och på följande sätt skapa samma lista med heltalen 1 till 10 som i exemplet med for ovan:

```
Tal = list(range(1,11))
```

Funktionen **`list()`** är alltså för typen Listor vad funktionen **`str()`** är för strängar och **`int()`** är för heltal.

Även additionsoperatorn **`+`** går att använda tillsammans med listor för att slå ihop dem: Följande kod gör att listan **`Tal3`** får innehållet **`[1, 2, 3, 4, 5, 6]`** :

```
Tal1 = [1, 2, 3]
Tal2 = [4, 5, 6]
Tal3 = Tal1 + Tal 2
```

Teori: Modulhantering

Till Python finns det flera olika moduler för att hantera slumptal, sannolikheter och statistik. En av dessa är modulen random som finns med i de flesta Pythoninstallationer. För att använda de funktioner som finns i ett bibliotek måste du först *importera* modulen. Ett sätt att göra detta är genom att i början av ditt program skriva:

```
import random
```

När du har gjort detta får du tillgång till de funktioner som finns i modulen random. När du ska använda funktioner i en modul som har importerats på detta sätt måste du ange modulens namn före varje funktion:

`x = random.randint(1,3)`	**x** blir ett slumpat heltal 1..3
`x = random.choice(Tal)`	**x** blir ett slumpat element ur listan **Tal**
`random.shuffle(Tal)`	blanda listan **Tal** slumpvis

För att skapa en lista med 10 st slumpvisa heltal mellan 1-100:

```
Tal = []
for _ in range(1, 11) :
    Tal.append(random.randint(1,100))
```

I ovanstående exempel används _ som stegvariabel i **for**-slingan. Det är ett sätt att indikera att man inte har något behov av stegvariabeln och därför utelämnar den.

Om man inte vill behöva skriva ut modulens namn innan varje funktion så kan man välja att importera alla funktioner från modulen:

```
from random import *

x = randint(1, 10)
```

Om man väljer att göra på detta sätt så måste man emellertid vara försiktig i sin kod så att man inte skapar en egen funktion med samma namn som en funktion i modulen du importerade. I så fall används den senast definierade funktionen, vilket vanligtvis är den egna.

Ett annat sätt att importera en modul för att det inte ska bli så långa uttryck är att ge den importerade modulen ett alias:

```
import random as rn

x = rn.randint(1, 10)
```

Om man ska använda en funktion i en modul ofta så kanske man till och med vill ge den ett alias:

```
from random import randint as ri

x = ri(1, 10)
y = ri(5, 17)
```

Övning 3

Denna övning kräver att du har gått igenom *Listor* och *Modulhantering*.

Du ska skriva ett program som kastar *n* stycken tärningar *x* antal gånger. Efter detta ska du kunna presentera hur många gånger summan av tärningarna har haft ett givet värde.

Utgå ifrån nedanstående skelettkod. Försäkra dig om att du förstår hur den är uppbyggd innan du går vidare och börjar arbeta med den.

```
from random import randint

n = int(input("Antal tärningar :"))
x = int(input("Antal kast:"))
dist = ...                      # Skapa svarslista
for _ in range(x) :             # Upprepa x antal gånger
    summa = 0
    for _ in range(n) :         # Upprepa för n tärningar
        summa = summa + ...     # Uppdatera tärningssumman
    dist[summa] = dist[summa] + 1
```

a) Slutför programmet ovan genom att komplettera de utelämnade delarna av koden markerade med `...`

Efter att du har kört ditt program så kan du studera resultatet genom att skriva **`dist`** vid prompten. Du får då se hur listan med fördelning av de olika summorna ser ut. I början av listan finns det alltid lika många nollor som antalet tärningar som har kastats (varför?).

Nu ska du få skriva en funktion som presenterar listan på ett lite mer läsbart sätt.

b) Skapa en funktion **`visadist(d)`** som tar en lista **`d`** med fördelning av tärningssummor som argument och presenterar den enligt följande:

```
Summan 2 vid xxx st kast
Summan 3 vid xxx st kast
Summan 4 vid xxx st kast
```

Inledande nollor i listan ska inte redovisas.

c) Komplettera ditt program från deluppgift a med ett anrop av funktionen **`visadist()`** för att visa resultatet efter att alla kast har genomförts.

I mån av tid:
För att plotta en graf över de värden som finns i en lista behöver du importera modulen pyplot i början av ditt program:

```
from matplotlib import pyplot
```

Efter detta kan du enkelt plotta resultatet av programmet i uppgift a) genom att i slutet av koden lägga till följande rader, som skapar en graf som sparas under filnamnet *plot.png*

```
pyplot.plot(dist)
pyplot.savefig('plot.png')
```

Övning 4

Om du vill arbeta med en textsträng tecken för tecken kan det underlätta att omvandla strängen till en lista. Följande kodexempel omvandlar strängen **`s`** till en lista med tecken som kallas för **`b`**:

```
b = list(s)
```

Som du redan sett så går en for-slinga igenom alla element i en lista. Tidigare har vi använt for i kombination med en lista skapad av funktionen **`range()`**, men det går precis lika bra att använda en godtycklig lista. För att gå igenom strängen **`s`** bokstav för bokstav kan du alltså skriva:

```
for x in list(s) :
```

En lista har en *uppräkningsbar* typ, vilket innebär att den har en längd. Detta innebär att du alltid kan få reda på antalet element i en lista genom att använda den inbyggda funktionen **`len()`**.

a) Ett palindrom är ett ord som blir likadant om man läser det baklänges. Skriv en funktion som tar en sträng som argument och returnerar **`True`** om strängen är ett palindrom.
Tips: Gör om strängen till en lista som du går igenom med en **`for`**-slinga. Du kommer också att ha nytta av funktionen **`len()`**.

b) Skriv en funktion som tar ett argument *n* och returnerar en lista innehållande *n* stycken slumptal mellan 1 och 100.

c) Importera modulen **`statistics`** och använd lämpliga funktioner i den för att beräkna median och medelvärde på en lista av slumptal skapad av din funktion i föregående deluppgift.
Tips: Konsultera online-dokumentationen för modulen genom att söka på nätet

d) För att lösa uppgift a) kan man även använda den inbyggda funktionen **`reversed()`**. Använd online-dokumentationen till Python för att lära dig hur denna funktion fungerar och skriv en lösning som avgör om en sträng är ett palindrom eller inte genom att använda funktionen **`reversed()`**.

Hemuppgift lätt: Programmering i matematik

Välj denna uppgift om du känner dig osäker på programmering och känner att du behöver arbeta lite mer med grunderna i programmering och Python.

- Ladda ner boken **Programmering i matematik** från http://oscillator.se/skola
- Arbeta igenom kapitel 3 (Python: introduktion)
- Välj ut ett projekt i kapitel 4 och genomför det.

  Om du valde ett projekt som det finns lösningsskiss till i kapitel 12: Jämför din lösning med din föreslagna och reflektera över skillnader.

- Välj ut ett projekt i kapitel 5 och genomför det.

  Om du valde ett projekt som det finns lösningsskiss till i kapitel 13: Jämför din lösning med din föreslagna och reflektera över skillnader.

Nästa träff

Nästa träff kommer vi att inleda med att tillsammans gå igenom och lösa den lite svårare hemuppgiften som presenteras på nästa sida. Om du vill förbereda dig inför detta kan du läsa igenom uppgiften så att du är insatt i vad den handlar om i förväg.

Hemuppgift svår: Yatzy

Välj denna uppgift om du vill ha en lite mer utmanande uppgift inför nästa träff. Vi kommer att inleda nästa träff med att genomföra denna uppgift tillsammans.

Skriv ett program som beräknar sannolikheten för att få Yatzy efter tre kast om du efter varje kast behåller de tärningar du har flest av. Utforma programmet så att du enkelt kan variera hur många omgångar à 3 kast som du gör för att räkna ut resultatet.

Vid lösningen av denna uppgift kan det vara bra att känna till hur man i en funktion arbetar med en variabel som är deklarerad utanför funktionen. Betrakta följande två exempel:

```
d = []

def doit(n) :
    d.append[n]
```

```
d = []

def doit(n):
    global d
    d.append[n]
```

Exemplet till vänster kommer inte att fungera eftersom en funktion inte kan modifiera en variabel som är deklarerad utanför funktionen. Lösningen är att göra som i exemplet till höger, där man i början av funktionen anger att variabeln **`d`** är definierad utanför funktionen med hjälp av nyckelordet **`globa`**l.

Några användbara tips:

- Håll reda på tärningarnas värden i en lista
- Det finns en funktion i modulen **`statistics`** som kan vara användbar för att hitta det värde som är vanligast förekommande i en lista
- Det går att räkna förekomsten av ett värde i en lista genom att använda funktionen **`count`**. Om du skriver **`d.count(x)`** kommer det att returnera hur många gånger **`x`** förekommer i listan **`d`**.
- Det kan vara en fördel att dela upp programmet i olika funktioner. Detta kan underlättas genom att börja med en skiss av hur programmet ska fungera, exempelvis:

  ```
  upprepa n gånger :
    upprepa 3 gånger :
      kasta tärningar
      spara tärningar
    kolla om yatzy
  ```

  etc. Obs! Ovanstående exempel är inte komplett.

När du har ett fungerande program, använd det för att beräkna sannolikheten för Yatzy och fundera på följande:

a) Hur många upprepningar måste du göra för att få en bra indikation av sannolikheten?

b) Den teoretiska sannolikheten för Yatzy efter tre kast är 3.43%. Det svar du har räknat fram avviker med största sannolikhet från detta. Varför?

Övning 1

a) != betyder "avskilt från", eller "inte lika med"

b) Svaret är ett booleskt värde, eller sanningsvärde (True/False)

c)
```
x = int(input("Ange ett tal:"))
if (isodd(x)) :
    print("udda")
else :
    print("jämnt")
```

d)
```
def isodd(x) :
    return ((x % 2) == 1)
```

e)
```
def collatz(n) :
    if isodd(n) :
        return n * 3 + 1
    else :
        return n / 2
```

f)
```
n = int(input("Ange n:")
while n > 1 :
    n = collatz(n)
    print(n)
```

Övning 2

a)
```
def find3s() :
    antal = 0
    for x in range(1000000,10000000) :
        if ("3" in str(x)) :
            antal = antal + 1
    return antal / 9000000
```

b)
```
def find3s(n) :
    antal = 0
    for x in range (10**(n-1), 10**n) :
        if ("3" in str(x)) :
            antal = antal + 1
    return antal / (9 * 10**(n-1))
```

Övning 3

a)
```
from random import randint

n = int(input("Antal tärningar :"))
x = int(input("Antal kast:"))
dist = [0] * (n*6 + 1)
for _ in range(x) :
    summa = 0
    for _ in range(n) :
        summa = summa + randint(1, 6)
    dist[summa] = dist[summa] + 1
```

b)
```
def visadist(d) :
    t = (len(d)-1) / 6   # Beräkna antal tärningar
    for i in range(t, len(d)-1) :
        print("Summan", i, "vid", d[i], "st kast")
```

c) Lägg till följande rad i slutet av programmet i a) :

```
visadist(dist)
```

Övning 4

a)
```
def palindrom(s) :
    svar = True
    l = list(s)
    for i in range(0, len(s)) :
        if l[i] != l[len(s)-i-1] :
            svar = False
    return svar
```

b)
```
def slumptal(n) :
    l = []
    for _ in range(n) :
        l.append(random.randint(1, 100))
    return l
```

c)
```
import statistics

lista = slumptal(1000)
print("Median :", statistics.median(lista))
print("Medelvärde :", statistics.mean(lista))
```

d)
```
def palindrom(s) :
    return list(s) == list(reversed(s))
```

LATHUND PYTHON

Tilldelning	
Tilldelning av variabel	`x = 7`
Tilldelning av lista	`lista = [1, 2, 3]`
Öka värdet på variabel	`x += 3`
Minska, multiplicera, dividera variabel	`x -= 3  x *= 3  x /= 3`
Omvandling av typer	
Skapa heltal	`int()`
Skapa flyttal (decimaltal)	`float()`
Skapa sträng	`string()`
Skapa lista	`list()`
Skapa mängd	`set()`
Operatorer	
Matematiska	`+ - * /`
Jämförelser	`== != < > <= >= is is not`
Logiska	`and or not`
Bitvis AND, OR, XOR, 1-komplement, Skift	`& | ^ ~ << >>`
Modulo, Heltalsdivision, Exponering	`% //`
Alternativ, IF-satser	
Enkel IF-sats	`if <villkor> :` `  ...`

IF-sats med ELSE	`if <villkor> :`
	`  ...`
	`else :`
	`  ...`

IF-sats med flera alternativ	`if <villkor> :`
	`  ...`
	`elif <villkor> :`
	`  ...`

IF-sats med flera alternativ och ELSE	`if <villkor> :`
	`  ...`
	`elif <villkor> :`
	`  ...`
	`else :`
	`  ...`

Slingor	
WHILE-slinga (styrs av villkorsuttryck)	`while <villkor> :` `  ...`
FOR-slinga (upprepning känt antal ggr)	`for <variabel> in <lista> :` `  ...`
FOR-slinga från a till b-1	`for <variabel> in range(a, b) :` `  ...`
FOR-slinga b gånger, från 0 till b-1	`for <variabel> in range(b) :` `  ...`

Lars-Åke Nordén
Uppsala Universitet

LATHUND PYTHON

Funktioner	
Definiera funktion utan argument	`def <funktionsnamn>() :` `    ...`
Definiera funktion med argument	`def <funktionsnamn>(<arg>) :` `    ...`
Returnera svar från funktion	`return <svar>`
Listor	
Skapa tom lista	`<lista> = list()`
Skapa tom lista	`<lista> = []`
Skapa lista med *n* stycken nollor	`<lista> = [0] * n`
Skapa lista med förifyllda värden	`<lista> = [1, 2, 3]`
Längd på lista	`len(<lista>)`
Lägg till nytt element sist i lista	`<lista>.append(<element>)`
Sortera lista	`sort(<lista>)`
Räkna förekomsten av värde i lista	`<lista>.count(<värde>)`
Min-värde i lista	`min(<lista>)`
Max-värde i lista	`max(<lista>)`
Element med givet index i lista	`<lista>[<index>]`
Slå ihop två listor till ny lista	`<lista> = <lista1> + <lista2>`
Kontrollera förekomst av värde i lista	`<värde> in <lista>`
Mängder	
Skapa tom mängd	`<mängd> = set()`
Skapa tom mängd	`<mängd> = {}`
Skapa mängd med förifyllda värden	`<mängd> = {1, 2, 3}`
Lägg till ett nytt element i mängden	`<mängd>.add(<element>)`

Funktioner som fungerar på samma sätt som för listor	`len(<mängd>)`
	`min(<mängd>)`
	`max(<mängd>)`
	`<värde> in <mängd>`

Användbara inbyggda funktioner	
Inmatning av text	`x = input()`
Inmatning av text med ledtext	`x = input("<ledtext>")`
Inmatning av heltal	`x = int(input())`
Inmatning av decimaltal	`x = float(input())`
Absolutvärdet av x	`abs(x)`
ASCII-värdet för tecknet x	`ascii(x)`
Tecknet med ASCII-värdet x	`chr(x)`
Avrundning av x	`round(x)`
Gör ingenting	`pass`
Funktionen `range()`	
Lista med värden från a till b-1	`range(a, b)`
Lista med värden från 0 till b-1	`range(b)`
Lista med värden från a till b-1 i steg om c	`range(a, b, c)`
Övrigt	
Inbyggd hjälpfunktion	`help()`
Hjälp om ett specifikt ämne	`help(<ämne>)`

Lars-Åke Nordén
Uppsala Universitet