Variabler, (data)typer, input, aritmetik

Med ett datorprogram behöver man kunna räkna, exvis de vanliga räknesätten: plus (+), minus (-), multiplikation (*) och division (/).

Ibland kan ej hela beräkningen göras på en gång, utan man måste räkna ut en sak i taget. Efter varje steg fås ett mellanresultat. Dessa mellanresultat måste sparas. För att kunna räkna behöver ett program använda s.k. *variabler*.

En variabel är som en låda i en ”byrålåda” där man kan lägga värdet (data) i. Byrålådan är jättestor, det är datorns minne. I varje låda kan lagras data av olika typer, exvis: heltal, decimaltal, text. Dessa lagras på lite olika sätt, men allt är binärt förstås. För att hålla reda alla data ger man varje byrålåda ett namn, det är variabelns namn.

I Python skapas en variabel genom att ge den ett värde. Detta kallas *tilldelning* och man gör detta med tilldelningstecknet =.

```
x = 4
```

Innebär att en variabel med namnet `x` har skapats och *tilldelats* värdet 4.

I liknelsen med byrålådorna, har det i en byrålåda lagts värdet 4 och lådans etikett (namn) är `x`. Eftersom värdet 4 tolkas som ett heltal kommer värdet lagras på det vis som heltal lagras. Man säger att *datatypen* (eller *typen*) för `x` är heltal (`int`)

En variabels namn får innehålla bokstäver, siffror och understrykningstecken, men får inte börja med siffra. Exvis: `xMax, sum, x2, antalTal,`

En god regel är att det börjar med en gemen och att namnet om möjligt säger lite om vad variabeln har för betydelse.

En variabels typ kan ändras, det beror på vad man tilldelar den.

Två exempel:

En variabel ”uppdaterar sig själv”:

```
x = 5
```
Förklaring: x tilldelas värdet 5

```
x = x*x
```
Förklaring: x ggr x räknas ut och tilldelas variabeln x, dvs 25

Byt värde på två variabler:

```
x = 5
y = 7
```

Idé:

```
x = y
```
Förklaring: x tilldelas värdet av y, dvs 7

```
y = x
```
Förklaring: y tilldelas värdet av x, dvs 7

Fungerar ej, ty 5 tappas bort.

Bättre:

```
t = x
```
Förklaring: t tilldelas värdet av y, dvs 5

```
x = y
```
Förklaring: x tilldelas värdet av y, dvs 7

```
y = t
```
Förklaring: y tilldelas värdet av y, dvs 5

Om man tilldelar en variabel ett matematiskt tal, ett s.k. *numerisk värde*, får den typen `int` (heltal) eller `float` (flyttal).

`t = 4`	typen `int`
`u = -6.32`	typen `float`: -6,32
`v = 1.14e31`	typen `float` (exponentform): $1{,}14 \cdot 10^{31}$
`x = 'Hej'`	typen `str`, sträng (text)
`y = '1234'`	typen `str`, sträng (text)
`z = math.pi`	typen `float`, ty `math.pi` är 3,1415…
`q = math.sqrt(7)`	typen `float`, √7 är 2.6457…

Ett flyttal kan skrivas med decimaler och/eller exponent.
Notera decimal anges med punkt, ej komma.
Math är ett ”bibliotek” med matematiska funktioner

Tilldelning har ett vänsterled och högerled

Högerledet räknas först ut…, sedan sker tilldelningen

```
x = 4         # x tilldelas värdet heltalet 4
y = x*9.5     # Högerledet, dvs x*9.5, räknas först ut…

4 = z       # kan ej skriva så  …
```

Tecknet # betyder att det till höger om # är kommentar

Läsa in värde till en variabel

Antag att när vi kör ett program skall följande hända:

```
Ge ditt namn: Kim
```

- Programmet frågar efter namnet, dvs skriver ut texten: `Ge ditt namn:`
- Programmet väntar på att användaren skriver något följt av return/enter.
- Användaren skriver `Kim` följt av return.
- Python-satsen som gör detta kan se ut så här:

```
namn = input('Ge ditt namn:')
```

`Input` är en inbyggd funktion i python som skriver ut en sträng (ledtext) och som därefter väntar på att användaren skall mata in något.
Variabeln `namn` tilldelas värdet av `input(…)` som är den text som användaren skriver. Värdet är alltid av typen `str` (sträng).
Variabeln `namn` är alltså av typen `str`.
Man kan säga att programmet (python) har frågat användaren efter en sträng och programmet har lagrat svaret i variabeln `namn`.

Antag att när vi kör ett program skall följande hända:

```
Ge ditt namn: Kim
Hej Kim
```

Dvs skriva ut texten `Hej` följt av det värde som användaren matat in tidigare.

- Python-satserna som gör detta kan se ut så här:

```
namn = input('Ge ditt namn: ')
print('Hej', namn)
```

`print` är inbyggd funktion i python för utskrifter. Med `print` kan man skriva ut text och variablers värden genom att ange ett antal argument som separeras med kommatecken. I vårt ex har vi två argument: Först argumentet som inramas med enkelblippar, dvs `'Hej',` tolkas som en text (sträng), som skrivs ut. Följt av argument utan enkelblippar, tolkas som en variabel vars värde görs om till text och skrivs ut. Mellan dem skrivs ut ett mellanslag.

Vad händer?

```
print('Hej', 'namn')
```

```
print(Hej, namn)
```

Läsa in numeriska data av typen float:

Eftersom input-funktionen alltid ger en sträng, måste den konverteras till numeriska data. Antag att när vi kör ett program skall följande hända:

```
Ge x: 9.0
Roten ur 9.0 är 3.0
```

- Programmet frågar efter ett värde.
- Programmet väntar på att användaren skriver något följt av return/enter.
- Användaren skriver `9.0` följt av return.
- Programmet skriver då ut texten: `Roten ur 9.0 är 3.0`
- Python-satserna som gör detta kan se ut så här:

```
s = input('Ge x: ')
x = float(s)              # konv från string till float
xrot = math.sqrt(x)       # math.sqrt är en funktion beräknar rotenur
print('Roten ur', x, 'är', xrot)
```

Satsen `x = floats(s)` konverterar strängen (`s`) till ett flyttal (`x`).

Vad händer om användaren matar in en text? Exvis: `Ge x: Kim`

Alternativ lösning, konverterar hela strängen direkt (rekommenderas):

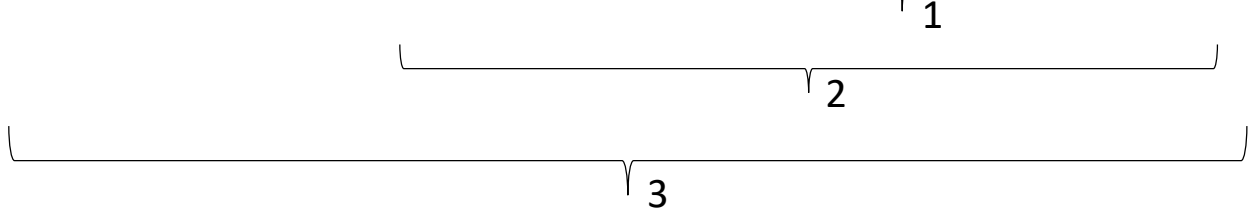
```
x = float(input('Ge x: '))
xrot = math.sqrt(x)
print('Roten ur', x, 'är', xrot)
```

Alternativ lösning:

```
x = float(input('Ge x: '))
print('Roten ur', x, 'är', math.sqrt(x))
```

Alternativ lösning (dock lite svårläst):

```
print('Roten ur x är', math.sqrt(float(input('Ge x: '))))
```



Läsa in numeriska data av typen int:

Python-satserna som gör detta kan se ut så här:

```
s = input('Ålder: ')
age = int(s)
```

Satsen `age = int(s)` konverterar strängen (`s`) till ett heltal (`age`).

Alternativt (rekommenderas):

```
age = int(input('Ålder: '))
```

Aritmetik, att räkna…

`+, -, * , /` fungerar som vanligt
Ett uttryck skrivs exvis `a+b` där kallas `a` och `b` för *operander*, `+` för *operator*.
Ett uttryck har inte bara ett värde, utan också en typ.
Om något av operanderna har typen `float`, blir uttrycket också `float`.,
undantaget / (division), där blir uttrycket alltid `float`. Exvis 4/5 = 0.8
* och `/` har högre prioritet än `+` och `-`
Parenteser för uttryck

```
a = 4
b = 5
c = 6 + a/b
print('c =' ,c)
```

→ `c = 6.8`

```
a = 4
b = 5
c = (6 + a)/b
print('c =' ,c)
```

→ `c = 2.0`

Utökade tilldelningsoperatorer

När man har en sats av typen: `a = a + 3`, dvs när en variabel skall uppdateras och dess värde står på båda sidor om tilldelningstecknet kan detta uttryckas:

```
a += 3
```

Det finns även operatorer för de andra räknesätten:

```
-=   *=    /=
```

```
a = 5
a += 2      Lika med a = a + 2, dvs 7
b = 6
b -= a      Lika med b = b - a, dvs -1
c = 7
c *= b      Lika med c = c*b, dvs -7
```

Andra aritmetiska operatorer `//` `%` `**`

Operatorn `//` utför division men ”kastar bort” decimalerna och ger resultatet närmaste lägre heltal.

`12 // 5` Blir 2 (ty 2 är närmast lägre 2.4)
`-5 // 8` Blir -1 (ty -1 är närmast lägre -0.625)

Operatorn % utför restberäkning (modulo):

`17 % 5` Blir 2, ty 17/5 = 3 + 2/5
`3 % 4` Blir 3, ty 3/4 = 0 + 3/4

Operatorn `**` utför upphöjt till:

`4 ** 2` Blir 16
`2 ** (-1)` Blir 0.5, ty 2**(-1)=1/2
`(-2) ** 3` Blir -8

Matematiska standardfunktioner

Det finns många inbyggda funktioner. Dessa finns samlade i en s.k. *modul* med namnet *math*.

```
import math
x = 9.0
y = math.sqrt(x)
print('Roten ur', x, 'är', y)
z = math.pow(x,3)
print(x, '^ 3 är', z)
```

```
Roten ur 9.0 är 3.0
```

```
9.0 ^ 3 är 729.0
```

För att få tillgång till funktionerna måste man göra: `import math`

Några av funktionerna:

`pi`	konstanten π = 3.141592
`exp(x)`	e^x
`log10(x)`	10-log av x
`sin(x)`	sin(x), x i radianer
`degrees(x)`	översätter grader till radianer
`radians(x)`	översätter radianer till grader
`max(x,y)`	största av x och y
`min(x,y)`	minsta av x och y
`abs(x)`	absolutvärdet av x

Alla funktioner finns beskrivna på https://docs.python.org/3/library/math.html

Modulen random

I den finns ett antal funktioner för att generera slumpmässiga tal.

```
import random
r = random.random()
```

0.528584591236779 9

```
dice = random.randint(1,6)
```

2

```
u = random.randint(1,35)
```

14

`random()` slumptal `x, 0≤x<1`

`randint(a,b)` slumpmässigt heltal `h, a≤h≤b`

Alla funktioner finns beskrivna på https://docs.python.org/3/library/random.html

Kommentering

Gör programmet mer lättläst och begripligt. Kan kommentera på två sätt:

Kortare, en raders kommentarer med #
Flera raders kommentarer med """

Exempel:

```
"""
Detta program frågar efter två heltal och byter värden
på dem samt skriver ut dem efter bytet
"""
tal1 = int(input('Ge ett första heltal:')) # Läs in talen
tal2 = int(input('Ge ett andra heltal'))
temp = tal1   # Spara ena värdet i temporär variabel
tal1 = tal2   # Byt värden
tal2 = temp
# Skriv ut resultat
print('Första talet = ', tal1, ' Andra talet = ', tal2)
```

Reserverade ord:
Språket Python version 3.7 består av 33 st reserverade ord (key words), dvs språkets ordförråd.

Identifierare:
Ett ord som du själv inför i ditt program, exvis ett variabelnamn.

```
x = 14      # x är en identifierare
```

För definierade datatyper:
int för heltal, *float* för flyttal, *str* för string, *bool* för bolska (dvs `True/False`)

Kontrollera vilken typ en variabel är:
`type(x)` → `<class 'int'>`

Kuriosa: Vilken minnesadress är en variabel lagrad i:
`id(x)` → 1371269248

Konvertering mellan datatyper (type casting):

```
x =  1234
s = str(x)          # s blir '1234'    (heltal → sträng)
y = int(s)+ 5       # y blir 1239 (sträng → heltal)

a = 3.6
b = int(3.6)        # b blir 3   (heltal), dvs trunkering…
c = 14
d = float(c)        # d blir 14.0   (heltal), dvs trunkering…
```

Notera konvertering görs bara om det är genomförbart:

```
s = '12x34'
x = int(s)  # ger felmeddelande
 ValueError: invalid literal for int() with base 10: '12x34'
```

Avrundning

Kan göras med funktionen round

```
 x = 7.456
print( round(x,1) )      # ger 7.5
print( round(x,2) )      # ger 7.46
print( round(x,0) )      # ger 7.0
```

Problem med flyttal

Flyttal kan ibland inte representeras exakt.
Heltal kan dock representeras exakt,
Exempel:

`a = 13`

`13 = 8 + 4 + 1 =` $1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 1101$ (binärt)

`x = 2.5 =` $1*2^1 + 0*2^0 + 1/2 = 1*2^1 + 0*2^0 + 1*2^{-1} = 10.1$ (binärt)

Binär pkt

`y = 2.56 =` $1*2^1 + 0*2^0 + 1/2 + 0.06 = 1*2^1 + 0*2^0 + 1*2^{-1} + 0.06$

Men vad är 0.06 uttryckt binärt?
Jo, $1/32 + e = 1*2^{-6} + e = 0.03125 + e$, där $e = 0.02875$.
Men vad är detta binärt? Jo ...