Python – Första steg

• LGMA20, L9MA20 • VT 2019 • Jonathan Nilsson, Laura Fainsilber •

Introduktion till Python

Programmering med Python

Python är ett populärt programmeringsspråk som används både inom industrin och på många utbildningar. Flera andra programmeringsspråk är också i bruk, som t.ex. Java, Matlab, och C++. Dessa är ungefär lika kraftfulla när det kommer till att göra effektiva matematiska beräkningar, men varje språk har sina för- och nackdelar. Den största fördelen med Python är att notationen är minimal så att koden blir lättläst. Python är dessutom gratis, har öppen källkod, program behöver inte kompileras, och man slipper hålla koll på krångliga saker såsom minneshantering eller typsättning av variabler.

Om man inte har programmerat tidigare kan det kännas svårt i början, det krävs helt enkelt lite tid för att vänja sig vid att skriva och tolka ett program. Det är också svårt att lära sig att programmera bara genom att läsa om det. Det bästa sättet är att prova sig fram och testa olika kommandon för att lära sig hur de fungerar. I instruktionen nedan har jag därför skrivit en guide till vad du kan prova att skriva in i Python för att komma igång. Det är viktigt att inte ha för bråttom när man går igenom texten. Tänk igenom hur de olika kommandona fungerar och försöka gissa vad resultatet kommer att bli innan du kör ett kommando. Var inte heller rädd för att göra dina egna små experiment med de olika operationerna för att se hur de fungerar.

Om man skriver ett helt program, och det inte fungerar kan det vara svårt att hitta felet. Det är därför bra att försöka köra små delar av programmet medan man skriver det - då ser man lättare när ett fel uppstår. Det är också en bra idé att försöka vänja sig vid att söka online efter svar på de frågor som dyker upp. En bra introduktion till Python finns på `https://www.w3schools.com/python/` - här kan man läsa om hur grundläggande kommandon fungerar. I den här kursen kommer vi dock i princip bara att behöva använda grundläggande räkneoperationer, if-satser, loopar, och funktioner.

Att använda denna text

Den här texten innehåller för tillfället en grundläggande introduktion till Python, med fokus på talteoretiska beräkningar. Sist i texten finns kursens datorlaborationer, och ett icke-obligatoriskt programmeringsprojekt för den som siktar på högre betyg. Jag rekommenderar att man försöker ta en titt på den grundläggande introduktionen på egen hand, på så vis har man mer tid att fokusera på uppgifterna under laborationerna. En uppgift om numerisk integration kommer att läggas ut i slutet på denna text, men du kan redan börja bekanta dig med Python.

Kom igång

Den här instruktionstexten är tänkt att användas tillsammans med *Anaconda* - en plattform för Pythonprogrammering. Anaconda finns installerat i datorsalen MVF22 som vi har bokad, men du kan också installera Anaconda på din egen dator: gå till `https://www.anaconda.com/download/`, ladda ned **Python 3.6** versionen, och följ installationsinstruktionerna (OBS! Programmet är en stor fil!). Applikationen som man startar programmeringsplattformen med kallas *"Spyder"*. I Windows 10 trycker man på Windowsfönstret längst ned till vänster och skriver "Spyder" för att hitta programmet.

I standardvyn i Spyder finns tre fönster. Till vänster finns en text-editor där man kan skriva in sin programkod. Nere till höger finns en "konsol" där man kan skriva in kortare kommandon, ungefär som på en vanlig miniräknare. I konsolen visas också in- och utdata när du kör dina program som du skrivit i det vänstra fönstret. Uppe till höger finns ett hjälpfönster där man snabbt kan få hjälp om hur man använder kommandon, man kan också i detta fönster växla till att se ett filsystem, eller till att se en lista över sina variabler.

Om du inte vill installera programmet utan skapa ett gratiskonto och köra direkt på internet kan du göra det på flera webplatser. Olika operativsystem och olika webläsare fungerar olika bra med olika varianter. Fönstren kan selite olika ut.

- `https://repl.it/languages/python_turtle`
- `https://jupyter.org`
- `https://eu.pythonanywhere.com`
- `https://trinket.io`
- `https://codeskulptor.org`
- `https://snakify.org`

Python som miniräknare

I konsolfönsret nere till höger står det `In [1]:` Här kan du skriva in kommandon som på en miniräknare och trycka enter för att köra dem. Operationerna + - * / fungerar precis som på en miniräknare. Parenteser fungerar också som vanligt, och decimalkomma skrivs med en punkt.

Python använder också lite notation som inte är så vanlig. "Upphöjt till" skrivs t.ex. som $**$ alltså med två multiplikationstecken. För att beräkna 2^{10} skriver man alltså `2**10`.

Genom att skriva $a//b$ beräknar du *heltalsdivisionen* av a med b, vilket är vanlig division fast avrundat nedåt. Det gäller t.ex. att $14//5 = 2$ och $1//2 = 0$.

Du kan enkelt beräkna resten vid heltalsdivision med Python med hjälp av operationen $\%$ (har ingenting att göra med procenträkning). För att beräkna resten när a divideras med b skriver du `a % b`. Vi har t.ex. $53\%7 = 4$ och $63\%9 = 0$.

Prova själv - räkneoperationer

Testa ovanstående kommandon tills du känner att du förstår dem. Utforska också följande:

- När behövs parenteser i Python? I vilken ordning beräknas 3+4*5 och 2**3**4 och 2*3%5?
- Vad händer om du försöker dela med noll?
- Hur snabbt är python? Prova att multiplicera två ungefär 20-siffror tal. Beräkna också 2^n för några stora värden på n. Var går gränsen för vad Python klarar?

Obs!

Ibland kan man råka skriva in nånting som skulle ta mycket lång tid för Python att beräkna. Du kan då stänga konsolen genom att klicka på det lilla röda krysset ovanför konsol-fönstret, då får du upp en ny konsol efter ett par sekunder.

För att få tillgång till fler matematiska funktioner i Python kan du skriva in följande rad i konsolen:

```
from math import *
```

Du har nu tillgång till flera matematiska funktioner. Du kan nu exempelvis beräkna roten ur 3 genom att skriva `sqrt(3)`. Du har också tillgång till trigonometriska funktioner, exponentialfunktionen, och logaritmer. Prova att skriva

```
cos(0)
sin(pi/6)
exp(1)
log(100)
log10(100000)
```

Variabler

Precis som på en miniräknare kan man skapa variabler och ge dem värden. Prova t.ex. att i konsolen skriva in

```
a=5
```

Du har nu skapat du en variabel a och tilldelat den värdet 5. Prova att på nästa rad skriva

```
a*a
```

Vad tror du händer om du härnäst skiver

```
a=a+1
```

Vilket värde har variabeln a nu? Prova genom att skriva `a` i konsolen.

Här ska man tänka på att "=" tecknet i Python används för att tilldela (ställa in) värdet på variabeln till vänster till det högra värdet. Att skriva `a=a+1` har alltså innebörden "sätt värdet på a till det aktuella värdet på a plus 1". Således har nu variabeln a värdet 6.

Variabler kan ha vilka namn som helst (utan mellanslag). Prova t.ex. att i konsolen i tur och ordning skriva följande:

```
hej=3
häst=7
apa=hej*häst
apa
```

I mer komplicerade program är det dock en bra idé att använda variabelnamn som påminner en om vad variabeln gör istället för djur. Variabler kan exempelvis heta "summan" eller "maxvärde" eller "svar".

Datatyper

Vi har hitills sett exempel på räkningar med heltal och decimaltal.

Strängar

Python kan också arbeta med text. En följd av tecken kallas en *sträng*. Strängar skrivs i Python inom citationstecken. Prova att i konsolen skriva in

```
x="Choklad"
y="muffins"
```

Du har nu skapat två variabel x och y som båda innehåller strängar. Strängar kan slås ihop med hjälp operationen +, eller upprepas med *. Prova att i konsolen skriva in följande. Gissa vad resultatet kommer att bli innan du trycker enter!

```
x+y
y+x
x*5
(x+y)*100
x*y
```

Växla mellan datatyper

Ibland vill man växla mellan olika datatyper. Du kan t.ex. konvertera ett decimaltal till ett heltal. Prova att skriva följande:

```
a=8.762
int(a)
round(a)
```

Prova själv - avrundning

Skriv i konsolen (om du inte redan gjort det)
`from math import *`
Du har nu också tillgång till funktionerna "floor" och "ceil" i konsolen.
Undersök vad skillnaden är mellan funktionerna floor, ceil, round, och int!

Ibland kan man få en sträng bestående av siffror - då behöver man omvandla strängen till ett heltal eller decimaltal för att kunna räkna med den numeriskt. Decimaltal kallas också flyt-tal, och man skriver float för att omvandla till flyt-tal. Prova att skriva in följande i konsolen:

```
s="3.1415"
s+1
s=float(s)
s+1
```

Från början är s en sträng, och vi kan inte addera ett heltal till en sträng. Men efter att vi gjort om s till ett decimaltal kan vi räkna med den som vanligt. Vad händer om du istället för float(s) skriver int(s)?

Det är också vanligt att man vill sätta ihop tal med strängar. Skriv i konsolen:

```
x=2019
"Det är nu år " + x +". Gott nytt år!"
```

Du får då ett felmeddelande. Detta beror på att du försöker slå ihop två olika datatyper: strängar och heltal. Detta kan lösas genom att konvertera variabeln x från ett heltal till en sträng innan vi slår ihop dem. Detta gör man genom att skriva $str(x)$. Skriv alltså istället:

```
x=2019
"Det är nu år " + str(x) + ". Gott nytt år!"
```

Att skriva program i Python

Från och med nu ska vi sluta använda konsolen, och istället skriva program i det vänstra fönstret. Börja med att skapa en tom fil och spara den någonstans, t.ex. på skrivbordet. När man skrivit ett program och sedan startar det genom att trycka F5 så kör man alla raderna kod efter varandra. Detta ger möjlighet att utföra mer komplicerade operationer. Man kan också lättare ändra sin kod och sedan köra hela programmet igen.

En skillnad mot att använda konsolen är att program inte ger någon output om man inte ber om det. För att få en utskrift från ett program måste man använda kommandot "print".

Prova att skriva följande enkla program i det vänstra fönstret, spara filen och kör den genom att trycka F5.

```
1 a=3
2 b=5
3 c=19
4 print("Produkten a*b*c blir " + str(a*b*c))
5 print("Summan a+b+c blir " + str(a+b+c))
```

Du kan sedan enkelt ändra värdena på a,b,c på rad 1-3 och köra programmet igen genom att trycka F5.

I ett program kan man be den som kör programmet att skriva in något genom att använda kommandot "input".

Skriv följande program i det vänstra fönstret, spara det och kör det genom att trycka F5.

```
1 x=input("Vad heter du? : ")
2 print("Du heter " + x + "! Vilket bra namn!")
```

Här använder vi kommandot "input" för att låta användaren ange en sträng x i konsolen. Sedan skapar vi en sammansatt sträng och skriver ut den med print-kommandot.

Obs!

Resultatet av input kommandot blir alltid en sträng. Om man vill att personen ska ange ett heltal eller decimaltal måste man alltså göra om strängen till en annan datatyp. Man kan t.ex. skriva

```
1 x=int(input("Ange ett heltal: "))
2 y=float(input("Ange ett decimaltal: "))
```

Då kan vi efteråt räkna med x och y som tal och inte som strängar.

Ibland glömmer man bort vad delar av ens kod gör. Därför är det bra att skriva *kommentarer* i Pythonkoden. En kommentar påverkar inte programmet alls. I koden skriver man

tecknet # följt av kommentaren på samma rad. Ovanstående program kan till exempel kommenteras såhär:

```
#Detta är ett enkelt program
x=input("Vad heter du? : ") #Be personen ange sitt namn
print("Du heter " + x + "! Vilket bra namn!") #Gratulera
```

Som du ser får kommandon, strängar, och tal olika färg i kodfönstret. Färgerna spelar ingen roll för programmet, men det hjälper en lättare läsa och tolka sin kod.

Loopar

Genom att använda **loopar** kan man upprepa delar av sin kod flera gånger, detta är mycket användbart som vi ska se framöver.

For-loopar

Den vanligaste typen av loop kallas för en for-loop. Testa t.ex. att köra följande program:

```
for x in range(1,8):
    print(x)
```

Som en mening kan programmet tolkas som "för varje heltal x i intervallet $[1, 8)$, skriv ut x". Programmet skapar alltså en loop-variabel x, och kör sedan den andra raden flera gånger där x först är 1, sedan 2, och så vidare.

Obs!

Notera att sista talet x som skrevs ut i föregående exempel är 7 och inte 8. Detta är en konvention i Python som man måste vänja sig vid: att när man skriver "x in range(a,b)" så är första värdet på x lika med a och det sista värdet för x är b-1.

Loopar kan t.ex. användas för att approximera summor och serier. Följande program beräknar de 100 första termerna i summan $\sum_{k=1}^{\infty} \frac{1}{k^2}$.

```
summa=0
for k in range(1,101):
    summa=summa+1/(k*k)
print(summa)
```

I steg nummer k adderar vi alltså termen $\frac{1}{k^2}$ till variabeln summa. Notera att vi var tvungna att sätta summa till 0 i början. (Varför?) Vad tror du händer om vi också skjuter in den fjärde raden i programmet?

Man kan också loopa igenom en lista med en for-loop:

```
frukter=["äpple","kiwi","banan","ananas"]
for f in frukter:
    print(f + " är gott!")
```

Här antar alltså loop-variabeln f i tur och ordning alla värden i listan frukter. Det spelar inte någon roll vad vi kallar denna variabel, den finns bara inne i loopen.

Prova själv - triangeltal

Ett triangeltal är ett tal på formen $1+2+3+\cdots+n$ för något n. Exempelvis är 3, 10, och 5050 triangeltal. Skriv ett program som skriver ut de första 100 triangeltalen! *Tips: detta går att göra med en enkel for-loop.*

Funktioner

Genom att skapa funktioner i Python kan man på ett enkelt sätt återanvända sin tidigare kod. Notationen för att definiera en funktion illustreras nedan, vi definierar funktionen $f(x) = x^2 + 3x + 5$ genom att skriva följande:

```
def f(x):
    return x*x
```

Funktionens namn är f, och stoppar man in x i funktionen så returneras x^2. Prova skriva in programmet och kör filen. Du får då ingen output, men funktionen f finns nu i datorns minne, och man kan räkna med den effektivt som med en vanlig funktion. Prova att direkt i konsolen skriva:

```
f(5)
f(f(0)+1)
häst=100
f(häst)
```

Det går också bra att skriva sådana rader i sitt program under definitionen av f, men då måste vi använda "print" kommandot om vi vill se utfallet.

Säg att vi vill räkna ut en Riemannsumma för arean under kurvan för $f(x)$ i intervallet [0,1], med 10 rektanglar. Varje rektangel är 1/10 bred och $f(x)$ hög, där x är i nedre vänsterhörnet av rektangeln, och vi tar summan av alla rektanglars area. Det kan vi göra genom att utöka ovanstående program såhär:

```
def f(x):
    return x*x

summa=0
for i in range (0,10):
        summa=summa+ (1/10)*f(i/10)
print(summa)
```

Här är det viktigt att definitionen av f står ovanför for-loopen: om inte f är definierad kan ju programmet inte beräkna f(i) inne i for-loopen.

En fördel med att göra på detta vis är att vi enkelt kan byta ut f mot en annan funktion i den övre definitionen och fortfarande använda den nedre koden. Testa med några fler funktionen som du kan integralen för, men även för $f(x) = exp(-x^2)$, (som beskriver normalfördelning) som inte är integrerbar.

En funktion kan heta vad som helst, kan ha hur många argument som helst (eller inga!), och kan innehålla hur många rader kod som helst innan den returnerar ett värde. Värdet som returneras kan också vara av vilken typ som helst. Funktionen behöver inte ens returnera något värde alls!

Här är ett exempel på en funktion som inte har några argument, och inte returnerar något värde:

```
def hälsning():
    for i in range(0,5):
        print("Ha en bra dag!")
```

Allt funktionen gör är alltså att skriva ut texten "Ha en bra dag!" fem gånger. Efter att vi definierat funktionen kan vi köra den genom att skriva

`hälsning()`

Om vi vill kunna variera hur många gånger programmet går genom loopen kan man göra en ny funktion, med en variabel:

```
def hälsningar(n):
    for i in range(0,n):
        print("Ha en bra dag!")
```

Kan du få programmet att skriva 10 hälsningar?

Här kommer ett exempel på en väldigt enkel funktion med två variabler, som beräknar skillnaden mellan två tal.

```
def jamfor(I,J):
      D=abs(J-I)
      print("Skillnaden mellan de två är "+ str(D))
```

Programmet jämför två tal, men du kan även använda den för att jämföra värdet av funktioner, t.ex genom att räkna "jamfor($f(10), f(11)$)".

Programmeringsuppgifter: Numerisk integration

Nu kommer du att programmera beräkning av integraler med hjälp av Riemannsummor. Skriv de program du behöver för varje uppgift. Spar filen (eller filerna) med ett bra namn. När du har kommit en bit visar du dina program för en lärare för examination. Du kan behöva köra programmet och förklara hur den funkar. Du kan få hjälp av kurskamrater och av lärare.

Uppgift 1: Summor med fler rektanglar

Använd programmet ovan (i avsnittet om funktioner) för att skriva ett nytt program som approximerar samma integral med hjälp av 100 rektanglar. Glöm inte att börja med "from math import *" och att definiera funktionen f. Testa ditt program med $f(x) = x^2$.
Bygg på din kod och definiera en ny funktion Riemann(n) som tar ett argument n och räknar (och skriver ut) en Riemannsumma för f med n rektanglar.
Tips: Se hur funktionen "hälsningar" är definierad.

Du kan nu jämföra det värde du får med 10, 100, 1000 rektanglar med $1/3$ (värdet av integralen $\int_0^1 x^2 dx$.

OBS! Du kan behöva lägga till

```
  return summa
```

i slutet av programmet så du får en output och inte bara en utskrift.

Du har nu en funktion som kan beräkna vänstersumma (dvs Riemannsumman där man använder värdet av funktionen i vänster hörnet av rektangeln) och skall skriva varianter.

Uppgift 2: Olika summor

Kopiera ditt program och modifiera det för att definiera en ny funktion Riemannhoger(n) som approximerar integralen genom att addera arean i n rektanglar vars höjd är värdet av funktionen i höger ändan av varje intervall.
Kopiera ditt program en gång till och definiera Riemannmitt(n) som använder värdet av f i mitten av varje intervall.

Du kan nu jämföra höger- och vänstersummor. För en växande (eller avtangande) funktion vet du att värdet av integralen ligger mellan höger- och vänstersumman.

Om du vill kan du även definiera en funktion Trapets(n) som använder trapetsmetoden (räknar areor av n trapetser vars övre sida är en sträcka från $(a, f(a))$ till $(b, f(b))$ om basen är intervallet $[a, b]$. Arean för en sådan trapets är $(b - a) \times (f(a) + f(b))/2$

Du har hittills approximerat integraler över intervallet $[0, 1]$ men skall nu utvidga din metod för att kunna variera högergränsen.

Uppgift 3: Summor från 0 till X

Kopiera ditt program och modifiera det för att definiera en ny funktion RiemannTillX(X, n) som approximerar integralen av f från 0 till X med n rektanglar. Glöm inte att justera bredden av rektanglarna.
Testa ditt program med funktionen $f(x) = x^2$ men även med andra funktioner vars integral du kan och med $f(x) = exp(-x^2)$, som du inte kan en primitiv funktion till.

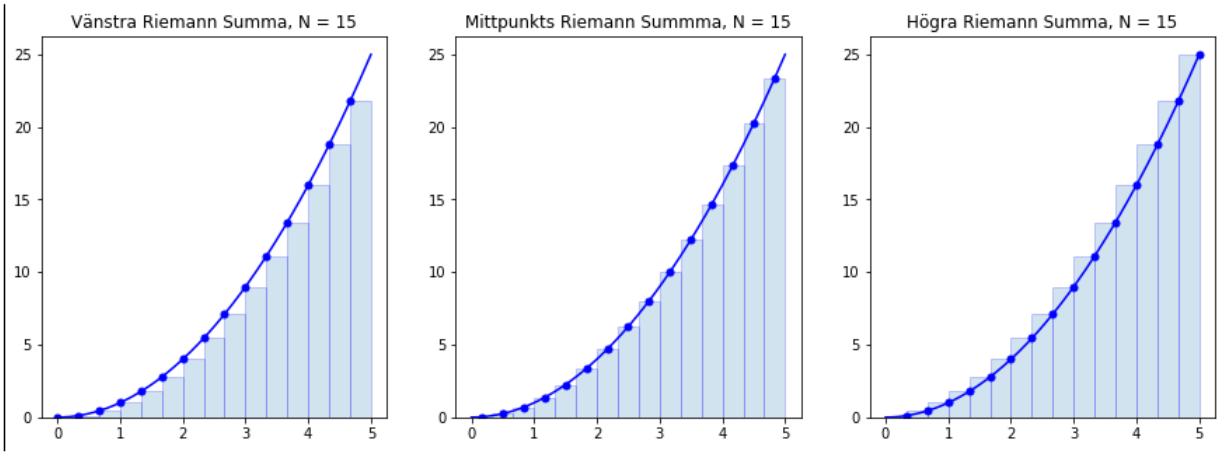
Att lägga till grafik är lite svårare, men här är bilder från ett program i Python som visar alla rektanglar för Riemannsumman för $f(x) = x^2$ mellan 0 och 5 med 15 rekanglar.

```
In [75]: runfile('C:/Users/laura/Downloads/fw.py', wdir='C:/Users/laura/Downloads')

Vad ska Punkten längst till vänster vara? 0

Vad ska punkten längst till höger vara? 5

Hur många intervall skall det vara? 15
Partition med 15 småintervall.
Vänstra Riemann Summa: 37.592592592592595
Mittpunkts Riemann Summa: 41.62037037037037
Högra Riemann Summa: 45.925925925925924
```



För dig som vill lära dig lite mer: If-satser och While-loopar

If-satser

Börja med att skriva följande program och testa vad det gör.

```
1 x=int(input("Ange ett tal: "))
2 if x>1000000:
3     print("Det var ett megastort tal!")
```

Detta är ett exempel på en if-sats. Koden på den inskjutna tredje raden körs endast om villkoret ovanför håller. Man kan också skjuta in fler rader efter varandra så att man får ett helt "block" med kod som bara körs om villkoret håller. Man skjuter in rader genom att använda tab-knappen (ovanför Caps Lock på tangentbordet). Efter if-satsen fortsätter programmet som vanligt.

Man kan också skapa if-satser med fler än ett fall i sina villkor:

```
1 x=int(input("Ange ett tal: "))
2 if x>1000000:
3     print("Det var ett megastort tal!")
4 elif x>1000:
5     print("Det var ett stort tal!")
6 elif x>10:
7     print("Det var ett lagom stort tal!")
8 else:
9     print("Det var ett litet tal!")
```

Här använder vi två nya kommandon. "Else if" skrivs elif, detta kan användas för att skapa fler fall i if-satsen. elif-villkoret på rad 4 testas endast om ovanstående fall inte inträffade. På samma vis testas villkoret på rad 6 endast om de föregående två fallen inte inträffade. På rad 8 står det bara "else:" utan villkor. Detta sista fall körs när *inga* av ovanstående fall inträffar.

Fundera på vad som hade hänt om vi hade byt ut "elif"-raderna mot vanliga "if"-satser och sedan skrivit in ett tal större än en miljon!

Här är ett mer praktiskt exempel. Följande program beräknar BMI-värden (ens kroppsvikt delat med kvadraten av ens längd i meter) och avgör om detta svarar mot undervikt/normalvikt/övervikt. Läs igenom och se om du förstå hur programmet fungerar.

```
1  L=float(input("Ange längd i cm: "))
2  V=float(input("Ange vikt i kg: "))
3  bmi=V/((L/100)**2)
4  print("BMI-värdet var " + str(bmi)+".")
5  if bmi<18.5:
6      print("Detta motsvarar undervikt")
7  elif bmi>=25:
8      print("Detta motsvarar övervikt")
9  else:
10     print("Detta motsvarar normalvikt")
```

Villkoret i if-satsen kan vara vad som helst av typen "sant/falskt", så det går bra att skriva komplicerade villkor. Om t.ex. x,y,z är tal och frukt är en lista med frukter så kan vi skapa if-satser som:

```
if (x>5 or y<0) and z==2:
```

eller

```
if "banan" in frukt:
```

Obs! "z==2" är en Boolesk variabel, dvs att den har två möjliga värde: "True" och "False".

Prova själv - jämnt eller udda

Skriv ett program som ber användaren att skriva in ett tal och sedan skriver ut i konsolen om talet är jämnt, eller udda. *Tips: Använd a%2 för att ta reda på om a är jämnt eller udda*

Mer om loopar

While-loopar

En annan typ av loop är while-loopen, denna låter oss upprepa ett stycke kod så länge ett villkor är sant. Följande program visar hur sådana kan användas:

```
1 x=1
2 while x<5000:
3     x=2*x
4     print(x)
```

Programmet fortsätter att dubbla värdet på x och skriver ut resultaten ända tills x blir större än 5000. Vilket blir det sista talet som skrivs ut? Varför? Vad händer om man byter plats på rad 3 och 4?

Om du vill krascha Python kan du skriva en *oändlig loop* av typen while(True), exempelvis

```
1 while 1==1:
2     print("Hjälp, jag är fast i en loop!")
```

Programmet fortsätter att köras så länge 1=1, det vill säga för alltid (eller i praktiken tills datorns minne tar slut eller tills du stänger Python-konsolen).

Loopar i loopar

Koden som körs innuti en loop kan vara vad som helst - exempelvis en annan loop!

Prova att skriva och köra följande program:

```
1 for i in range(1,5):
2     for j in range(1,4):
3         print(str(i) + " och " + str(j))
```

Tänk igenom hur programmet fungerar, lägg speciellt märke till den dubbla rad-inskjutningen på rad 3. Hur många utskrivna rader fick du? Lägg också märke till ordningen av utskrifterna.

I följande program använder vi ännu en dubbelloop för att skriva ut en multiplikationstabell av storlek 10*10.

```
1 for rad in range(1,11):
2     x=""
3     for kol in range(1,11):
4         x=x+str(rad*kol)+" ")
5     print(x)
```

Här upprepas rad 2-5 tio gånger medans variabeln rad går från 1 till 10. För varje värde på rad börjar vi med att skapa en tom sträng x, och sedan låter vi variabeln kol gå från 1 till 10 medans vi lägger till rad*kol till strängen x. Tillslut skriver vi ut raden x. Programmets fjärde rad kommer alltså att köras totalt 100 gånger. Du kan enkelt göra en större tabell genom att öka talet 11 i range-kommandot.