Programmera i matematik

Att använda programmering som ett verktyg i matematikundervisningen i årskurs 7-9

av

Staffan Melin

Tack till

Marino Sanvincenti
Mikael Bonnier
Lars-Åke Nordén
Göran Fagerström
Leif-Jöran Olsson
Edward Krogius
Martin Müntzing
Daniel Pamp
Mina elever på Bild & form-skolan i Göteborg

Bokens hemsida: oscillator.se/skola

Boken har producerats med fria verktyg:
GNU/Linux Debian, LibreOffice, IDLE, GIMP och Inkscape.

Version
2021-12-25

1. Förord

Syftet med den här boken är att elever i årskurs 7-9 ska arbeta med programmering som ett verktyg i matematikundervisningen.

Som språk har jag valt Python 3, ett av världens vanligaste programmeringsspråk, och ett språk som går att använda till det mesta.

Boken förutsätter inga förkunskaper i Python, och inte heller i programmering. Men det är bra om du som lärare samt eleverna har provat på någon form av programmering tidigare, exempelvis Hour of Code (https://code.org).

Ibland är inte programmeringen i boken den mest effektiva -- men kanske den tydligaste.

Hör gärna av dig om du hittar något fel i boken.

Staffan Melin

staffan.melin@oscillator.se
Legitimerad lärare i matematik, fysik och teknik i grundskolan
Civilingenjör
Utbildad journalist
Författare

2. Inledning

Om du är lärare börja då med kapitel 11 - Lärardel. Där får du tips på hur du kan arbeta med boken samt förslag på planering.

Boken börjar med en introduktion till Python. Huvuddelen av boken utgörs av olika elev-projekt som är indelade efter matematiskt område.

I slutet av boken ges förslag till lösningar, länkar till andra resurser samt lite tekniska tips.

I boken skrivs Python-kod alltid på grå bakgrund, exempelvis:

```
>>> 4 + 5
9
```

De tre större-än-tecknena (>>>) ska inte skrivas in. De anger att koden ska skrivas in interaktivt. Siffran 9 skrivs ut av Python.

Även hela program ligger på grå bakgrund, exempelvis:

```
a = int(input("Skriv in ett tal mellan 1 och 10:"))
if a == 5:
    print("Talet är fem!")
```

Eftersom koden inte börjar med >>> ska den skrivas in som ett program.

All kod bör skrivas in och köras när eleverna arbetar med projekten.

Boken är baserad på Python 3. Läs gärna appendixet om programmeringsmiljö om du är nybörjare.

Innehåll

3. Python: Introduktion

3.1. Att köra programkod

Det finns många olika sätt att skriva in och köra Python-kod: appar, installerade program, i webbläsaren.

I denna bok använder vi oss av Python 3.

Se Appendix: Programmeringsmiljöer.

3.2. Rad för rad: Datorn som miniräknare

3.2.1. Uttryck

Python är ett interpreterande språk. Det betyder att det tolkar språket rad för rad direkt när Python läser det.

Det innebär också att du direkt kan börja använda Python som en miniräknare om du använder det interaktivt.

Pröva att skriva in följande matematiska uttryck:

```
>>> 4 + 5
9
>>> 4 + 6 + 3
13
>>> 4 + 6 * 3
22
>>> (4 + 6) * 3
30
```

Som du ser använder vi en stjärna (*) som multiplikationstecken.

Python använder sig av samma prioriteringsregler som matematiken:

1. () parenteser
2. */ multiplikation och division
3. +- addition och subtraktion

Du kan också dividera tal

```
>>> 6 / 2
3.0
>>> 8 / 4
2.0
```

Men vad händer om bråket inte går jämnt ut?

```
>>> 17 / 3
5.666666666666667
```

Vi får ett decimaltal. Det kallas float, till skillnad från heltal som kallas för int (integer).

Vad blir då:

```
>>> (17 / 3) * 3
17.0
```

Om vi vill ha heltalsdelen av resultatet från en division så skriver vi:

```
>>> 17 // 3
```

```
5
```

Vi kan också se vad resten blir i en division genom att skriva:

```
>>> 17 % 3
2
>>> 15 % 2
1
```

Det kallas för modulus.

Vi kan också räkna med exponenter:

```
>>> 3 ** 2
9
```

Det vill säga 3 · 3 ("3 gånger sig själv 2 gånger").

```
>>> 2 ** 4
16
```

Det vill säga 2 · 2 · 2 · 2.

Det är alltid en bra idé att använda parenteser för att förtydliga ordningen av dina beräkningar, även om de i Python följer de vanliga matematiska prioriteringsreglerna.

```
>>> 10 - 4 - 2
4
>>> (10 - 4) - 2
4
>>> 10 - (4 - 2)
8
```

1. Vi har en grupp med 4 elever och 2 lärare. Alla köper 4 lakritsbitar och 12 surisar var. Hur räknar du ut hur många godisbitar som gruppen köper totalt? Du ska skriva det som 1 uttryck.

2. Vilken prioritet har **? Beskriv hur du kan ta reda på det. Tips! Gör olika beräkningar där du använder dig av ** tillsammans med andra räknesätt.

3.2.2. Variabler

Alla programmeringsspråk har **variabler**. De kan ses som lådor där vi kan stoppa olika värden. Alla lådor har ett namn, variabelns namn.

I Python, liksom i många andra programmeringsspråk, är = inte ett likhetstecken utan ett tilldelningstecken. Det används för att tilldela, alltså ge, en variabel ett värde. Du "stoppar in ett värde i lådan".

```
>>> vikt = 4
>>> antal = 3
```

Det finns några regler för vilka namn en variabel kan ha. Namnet kan bara bestå av:

- bokstäverna i alfabete, alltså a till z, samt åääö
- understrykningstecken (_)
- siffror (men variabelnamn kan inte börja med en siffra)

Det innebär att följande är godkända variabelnamn: vikt, antal, antal_siffror, bil1, bil2.

När det gäller variabelnamn så är a och A olika bokstäver. Variablerna antal och Antal är alltså två olika variabler. I denna bok använder vi gemena bokstäver för variabelnamn. Se Pythons stilguide (Appendix: Kodstil).

Om du vill använda ett variabelnamn som ska bestå av två eller flera ord så skriver du ett understrykningstecken mellan varje ord. Exempel: antal_siffror, antal_gula_bilar.

Det finns olika åsikter om det är bra eller dåligt att använda de svenska tecknena åääö i variabelnamn. I denna bok gör jag inte det eftersom det inte brukar vara tillåtet i andra programmeringsspråk.

Vill du ha reda på vad en variabel har för värde använder du dess namn.

```
>>> vikt
4
```

Du kan också använda variabler i matematiska uttryck:

```
>>> antal * vikt
12
>>> antal = 4
>>> antal * vikt
16
```

Vi kan dessutom tilldela två variabler var sitt värde på samma rad:

```
>>> antal, vikt = 5, 4
>>> antal * vikt
20
```

Vi kan också tilldela en variabel värdet av en annan variabel:

```
>>> antal_applen = 4
>>> antal_paron = antal_applen
>>> antal_frukter = antal_applen + antal_paron
```

1. Skapa variabler för den fysikaliska formeln sträckan = hastigheten gånger tiden.

2. Tilldela hastigheten och tiden olika värden. Hur räknar du ut sträckan med hjälp av variablerna och dess värden?

3.2.3. Text

En annan typ av variabel kan innehålla text. De kallas för sträng-variabler eller text-variabler.

```
>>> a = "ma"
>>> b = "te"
>>> c = a
>>> d = "tik"
>>> a + b + c + d
'matematik'
```

Som du ser ska text som du anger omslutas av ". Du kan också använda '. Detta är inte ett citationstecken, även om det kan se ut som det. Citationstecken är ofta lite ”krulliga”. När du skriver in det trycker du dock Shift+2 som vanligt. Plustecknet (+) slår ihop text.

Om du behöver använda dig av " inuti din text använder du \":

```
>>> prat = "Charlie sa \"Hej\" till Kim."
```

Vi kan dessutom upprepa en text-variabels värde genom *-tecknet

```
>>> (3 * a) + b
'mamamate'
```

Om vi vill plocka ut en bokstav från en text-variabel så skriver vi

```
>>> a[0]
'm'
>>> a[1]
'a'
>>> a + b[0] + b
'matte'
```

0 är alltså första tecknet i variabelns värde. I programmeringssammanhang börjar vi nästan alltid räkna från 0, inte 1.

```
>>> a[-1]
'a'
```

Med ett minustecken hämtar Python talet från slutet av variabelns värde.

Vi kan dessutom plocka ut flera bokstäver:

```
>>> ord = a + b + c + d
>>> ord[2:6]
'tema'
```

Det vill säga, bokstav 2 till 6 (men inte *till och med* 6). Kom ihåg att vi räknar från bokstav 0.

```
>>> ord[3:]
'ematik'
```

Om vi inte anger någon siffra efter kolon så plockas tecken ända till slutet.

Ibland vill vi veta hur lång en sträng är. Då använder vi:

```
>>> len(ord)
9
```

len() kallas för en funktion. Du kommer att lära dig mer om funktioner senare.

1. Lägg in ditt förnamn i en varibel med namnet ”fornamn”. Lägg in ditt efternamn i en variabel med namnet ”efternamn”. Skapa en variabel med namnet ”initialer” och lägg in första bokstaven från ditt för- och efternamn.

3.2.4. Decimaltal

Variabler kan också innehålla decimaltal.

```
>>> c = 1/2
>>> c
0.5
```

Python använder en punkt som decimaltecken vilket är standard i det engelska språket.

Du kan omvandla decimaltal till heltal med hjälp av funktionen int().

```
>>> d = int(c)
>>> d
```

När du gör om dem till heltal avrundas de neråt vilket sällan är vad du vill. I matematiken avrundar vi ju normalt tal med siffran 5 på slutet uppåt.

Python har en inbyggd funktion, round(), men inte heller den fungerar som vi är vana vid (den avrundar till närmast jämna tal om talet slutar på 5).

Prova istället att lägga till 0,5 till talet (vilket bara fungerar för positiva tal):

```
>>> d = int(c + 0.5)
>>> d
```

Du kan göra om heltal och decimaltal till text genom att använda funktionen str(). Detta kan vara användbart om du vill sätta ihop talet med text.

```
>>> a = 5
>>> utskrift = "Variabeln a har värdet " + str(a)
>>> utskrift
'Variabeln a har värdet 5'
```

1. Vad ska du lägga till ett negativt tal för att avrundning med int() ska fungera som du lärt dig på matematik-lektionerna?

3.3. Program: Inmatning och beslut

Hittills har vi bara använd Python som ett språk som utför en rad i taget.

Vi vill dock kunna skriva flera rader som hänger ihop. De kallas för program.

Det blir dessutom både intressantare och mer användbart om vi skriver program som kan utföra olika saker beroende på vilken information de får.

3.3.1. Imatning av tal

Vi börjar med en möjlighet för programmet att hämta in information från användaren. Den funktionen heter input(). Skriv in följande program och kör det.

```
a = input("Skriv in ett tal:")
print(a)
```

Pröva nu:

```
a = input("Skriv in ett tal:")
print(a * 2)
```

Säg att du skriver i talet 9. Du kommer att få svaret 99, inte 18 som du antagligen förväntar dig. Det beror på att input läser in allt som text.

Skriv därför istället så här om du vill mata in ett tal:

```
a = int(input("Skriv in ett tal:"))
print(a)
print(a * 2)
```

Genom att omge input() med int(), så omvandlas svaret från input till ett heltal (integer).

Vi använder funktionen print() för att skriva ut värden.

```
a = 5
print(a)
a = "fem"
print(len(a))
```

Du kan också skriva ut flera värden genom att skriva ett kommatecken mellan dem.

```
a = 1
b = 2
print(a, b)
```

3.3.2. Jämförelser och indrag

Om vi vill undersöka vilket värde som en variabel har kan vi använda oss av if:

```
a = int(input("Skriv in ett tal mellan 1 och 10:"))
if a > 5:
    print("Du skrev in talet", a)
```

```
    print("Talet är större än 5!")
```

När du skrivit in en rad som avslutas med kolon (:) så ser du att nästa rad är förskjuten till höger. Det kallas för att den är indragen. Indragna rader är Pythons sätt att gruppera rader i block. Ett indrag görs normalt med fyra stycken mellanslag. **Alla rader som ska köras om a är större än 5 måste ha samma indrag.** I exemplet ovan innebär det att båda print()-raderna utförs om a är större än 5 eftersom de har samma indrag. Om a är mindre än eller lika med 5 skrivs inget ut.

Vi kan också undersöka om ett värde är lika stort som ett tal. Då använder du == som betyder ”lika med”.

```
a = int(input("Skriv in ett tal mellan 1 och 10:"))
if a == 5:
    print("Talet är fem!")
```

Vi kan också ange vad som ska hända om vårt test *inte* är sant. Då använder vi else.

```
a = int(input("Skriv in ett tal mellan 1 och 10:"))
if a == 5:
    print("Talet är 5!")
else:
    print("Talet är inte 5!")
```

Men det går också att kontrollera genom !=, "inte lika med":

```
if a != 5:
    print("Talet är 5!")
```

Vi kan också göra flera jämförelser i rad med hjälp av elif (”else if”).

```
alder = int(input("Skriv in din ålder:"))
if alder > 65:
    print("Du är pensionär!")
elif alder > 18:
    print("Du är vuxen!")
elif alder > 12:
    print("Du är tonåring!")
else:
    print("Du är ett barn!")
```

Vi kan också göra flera jämförelser i samma test genom att använda oss av or och and (eller och och).

```
manad = int(input("Skriv in numret på din födelsemånad:"))
if manad == 12 or manad < 3:
    print("Du är född på vintern!")
elif manad >= 3 and manad <= 5:
    print("Du är född på våren!")
```

>= betyder "större än eller lika med". <= betyder "mindre än eller lika med".

1. Skriv ett program där du ska mata in ett tal. Programmet ska skriva ut kvadraten av talet (det vill säga talet multiplicerat med sig själv) med hjälp av **.

Vi kan också göra indrag i flera nivåer.

```
a = int(input("Skriv in ett tal mellan 1 och 10:"))
if a > 9:
    print("Talet är 10 eller större!")
    if a > 99:
        print("Talet är 100 eller större!")
```

I det här fallet kommer jämförelsen om talet a är större än 99 endast göras om a redan är större än 9, vilket ju är rimligt.

3.3.3. Inmatning av text

Du använder input() även till inmatningar av strängar, det vill säga text.

```
namn = input("Vad heter du?")
print("Du heter", namn)
```

Vi kan också göra jämförelser av strängar.

```
namn = input("Vad heter du?")
if namn == "Anna":
    print("Du har Sveriges vanligaste tilltalsnamn!")
elif namn == "Lars":
    print("Du har Sveriges vanligaste tilltalsnamn!")
```

Dessa två jämförelser kan slås samman till en genom att använda or (eller).

```
namn = input("Vad heter du?")
if namn == "Anna" or namn == "Lars":
    print("Du har Sveriges vanligaste tilltalsnamn!")
```

Du kan också använda dig av and (och) på samma sätt som för tal.

1. Skriv ett program där du ska mata in ett ord. Programmet ska sedan skriva ut om ordet är kort (färre än 5 bokstäver) eller långt (fler än 4 bokstäver).

2. Skriv ett program där du ska mata in ett hundnamn. Programmet ska sedan ta reda på om namnet är ett av de två vanligaste hundnamnen.

2. Skriv ett program där du ska mata aktuell månad och dag. Programmet ska sedan skriva hur många dagar som gått sedan årsskiftet.

3.4. Slingor

Om vi vill skriva ut alla heltal från 1 till 100 så kan vi antingen skriva ut dem ett och ett:

```
print(1)
print(2)
```

och så vidare upp till

```
print(100)
```

Det blir många rader! Denna upprepning gör vi enklare med hjälp av något som kallas för slinga. En slinga är ett antal programrader som upprepas ett visst antal gånger.

Det finns olika sätt att skapa slingor i Python. De två vanligaste använder sig av while och for. While utförs så länge ett uttryck är sant medan for utförs ett förutbestämt antal gånger.

Du kommer att lära dig mer om slingor i Område: Tal.

3.4.1. While

Med while (engelska: medan) skriver vi ut talen från 1 till 100 på detta sätt:

```
tal = 1
while tal <= 100:
    print(tal)
    tal = tal + 1
```

Vi har här infört variabeln tal som fungerar som en räknare. En räknare är en variabel som hela tiden ökar sitt värde.

Raden while tal <= 1000 utför ett test och så länge det testet stämmer så utförs programraderna som följer och är indragna. Dessa två indragna rader skriver ut talets värde och ökar sedan räknaren med 1.

1. Skriv ut alla tal från 1000 till 1111 med hjälp av while.
2. Skriv ut alla jämna tal från 0 till 1000 med hjälp av while.

3.4.2. For

Vi kan också skapa slingor med for och range().

```
for tal in range(1, 10):
    print(tal)
```

Detta program skriver ut alla tal från 1 till 9. Det inkluderar alltså inte sluttalet.

Det börjar med att skapa en "range", en mängd med tal. Inom parentesen anger vi var vår mängd startar och slutar.

for-satsen gör sedan att variabeln tal i tur och ordning kommer att tilldelas värdena i denna

mängd. Så först är tal lika med 1, sedan 2, 3, 4 och så vidare ända till tal är lika med 9.

1. Ändra programmet så att det skriver ut alla tal från 1 till 10.

3.5. Funktioner

3.5.1. Enkla funktioner

Vi har tidigare använt oss av något som kallas funktioner, exempelvis len() och str().

Funktioner är programkod som vi eller någon annan skrivit och som vi kan använda flera gånger.

- len() är en funktion som tar reda på hur lång en text är.
- str() är en funktion som omvandlar ett sifferuttryck till text.

Båda dessa funktioner finns inbyggda i Python.

Men vi kan också skriva egna funktioner för programrader som vi vet att vi kommer vilja återanvända flera gånger.

Om vi vill ta reda på vilket av två tal som är störst så kan vi lägga det testet i en funktion. Då kan vi använda oss av den funktionen på flera ställen i vårt program.

```
def jamforTal(a, b):
    if a > b:
        print("Det första talet är störst!")
    elif a < b:
        print("Det andra talet är störst!")
    else:
        print("Talen är lika stora!")
```

Vi har nu skapat en funktion kallad jamforTal (jämförTal -- men i denna bok använder vi inte svenska tecken i funktions- och variabelnamn) som tar två tal som argument, alltså in-data. Observera kolonet och de följande indragen efter funktionens första rad.

Vi använder funktionen så här:

```
jamforTal(16, 15)
```

Då får a värdet 16 och b värdet 15 och sedan körs funktionens programkod.

1. Skriv en funktion som jämför två tal och talar om vilket av dem som är minst.

2. Skriv en funktion som undersöker vilket av två namn som är längst. Kom ihåg att du kan använda dig av den inbyggda funktionen len() för att ta reda på hur långt innehållet i en variabel är.

3.5.2. Funktioner med returvärden

Nu ska vi skriva en syskon-funktion till jamforTal() kallad storstaTal() (störstaTal). Den tar två tal som argument och skickar tillbaka det största av de två. Att skicka tillbaka ett värde

heter på programmeringsspråk att den returnerar ett värde. Kom ihåg att funktionen len() skickade tillbaka längden på innehållet i strängen som vi skickade till funktionen!

Vi vill alltså kunna använda den så här:

```
tal = storstaTal(200, 250)
print("Det största talet var", tal)
```

Så här ser funktionen ut:

```
def storstaTal(a, b):
    if a > b:
        return a
    elif a < b:
        return b
    else:
        return b
```

Vi kan korta ner funktionen:

```
def storstaTal(a, b):
    if a > b:
        return a
    else:
        return b
```

1. Skriv en funktion som tar två tal som argument och skickar tillbaka det minsta.

2. Skriv en funktion som tar två namn som argument och skickar tillbaka längden hos det längsta.

3.6. Projekt: Läxförhör

Skriv ett läxförhörsprogram som prövar användarens kunskaper med hjälp av 3 frågor. Frågorna kan antingen gälla numeriska svar (alltså tal) eller text (strängar).

Skapa en variabel som håller reda på hur många rätt användaren har. Om användaren gissar rätt ska du öka variabelns värde med 1. Efter de tre frågorna ska du skriva ut hur många rätt användaren hade.

4. Område: Tal

4.1. Projekt: Multiplikation och division

Multiplikation av heltal kan ses som en upprepad addition. Exempelvis är $3 \cdot 5 = 5+5+5$. Nu ska vi skriva ett program som beräknar produkten av två tal med hjälp av addition.

```
tal1 = int(input("Det första talet:"))
tal2 = int(input("Det andra talet:"))
produkt = 0
```

Vi vill nu kunna lägga till tal2 till produkten lika många gånger som anges av tal1. Så om vi matar in 3 och 5 så kommer variabeln produkt vara 5+5+5. Svaret lägger vi i variabeln produkt.

Vi behöver en räknare som håller reda på hur många gånger vi utfört additionen.

```
raknare = 0
```

Sedan använder vi oss av while och hela programmet ser ut så här:

```
tal1 = int(input("Det första talet:"))
tal2 = int(input("Det andra talet:"))
produkt = 0
raknare = 0
while raknare < tal1:
    produkt = produkt + tal2
    raknare = raknare + 1
print(produkt)
```

Vi kan lättare se hur slingar arbetar om vi lägger in en utskrift i slingan med hjälp av print():

```
tal1 = int(input("Det första talet:"))
tal2 = int(input("Det andra talet:"))
produkt = 0
raknare = 0
while raknare < tal1:
    produkt = produkt + tal2
    raknare = raknare + 1
    print("Produkt=", produkt)
print(produkt)
```

1. Potenser kan ses som en upprepad multiplikation. Exempelvis är $2^3 = 2 \cdot 2 \cdot 2$. Skriv ett program som räknare ut $tal1^{tal2}$.

2. Skriv ett program som räknar ut division av heltal genom upprepad subtraktion.

Du kan exempelvis räkna ut 16/3 genom att se hur många gånger du kan subtrahera 3 från 16:

16 - 3 = 13 (1 gång)

13 - 3 = 10 (2 gånger)

10 - 3 = 7 (3 gånger)

7 - 3 = 4 (4 gånger)

4 - 3 = 1 (5 gånger)

Nu är resten mindre än nämnaren, så det går inte att subtrahera fler gånger. Svaret blir då att 16 / 3 är lika med 5 med resten 1.

4.2. Projekt: Jämna och udda tal

Nu ska vi undersöka några olika egenskaper hos tal.

Vi börjar med att lära oss hur vi skapar en lista med tal.

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

Detta program skriver ut alla tal från 1 till 10.

Det börjar med att skapa en "range", en mängd med tal. Inom parentesen anger vi var vår mängd startar och slutar. Men range inkluderar inte sluttalet, därför ska vi ange 11.

for-satsen gör sedan att variabeln tal i tur och ordning kommer att tilldelas värdena i denna mängd. Så först är tal lika med 1, sedan 2, 3, 4 och så vidare ända till tal är lika med 10.

För att skapa jämna tal kan vi ta vilket tal som helst och multiplicera det med 2. Resultatet blir alltid jämnt. Kan du svara på varför det är så?

```
>>> tal = 3
>>> print(tal * 2)
6
>>> tal = 5
>>> print(tal * 2)
10
>>> tal = 17
>>> print(tal * 2)
34
```

De jämna talen är oändligt många så om vi vill skriva ut dem så måste vi begränsa oss.

Vi bestämmer oss för att skriva ut alla jämna tal upp till 100. Varför skapar vi en range() som endast går upp till 51?

```
for tal in range(1, 51):
    print (tal * 2)
```

Vi kan även skapa jämna tal med hjälp av while. Vi gör det genom att skapa en variabel, låta den börja med att vara 1 och därefter i varje steg öka dess värde med 2.

```
tal = 2
while tal < 100:
    tal = tal + 2
    print(tal)
```

För att skapa udda tal kan vi även använda oss av en extra egenskap hos range(). range() kan ta ett extra värde (på programmeringsspråk kallat för argument) för hur mycket stegen ska öka.

```
for tal in range(1, 101, 2):
```

```
    print(tal)
```

1. Skriv ut alla tal från 100 till 199 med hjälp av for.

2. Skriv ett program som listar alla udda tal från 1 till 101 med hjälp av for.

3. Skriv om programmet så att det skriver ut alla udda tal från 1 till 101 med hjälp av while.

4. Skriv ett program som skriver ut alla tal som är delbara med 3 från 1 till 100. Använd först for och därefter while.

4.3. Projekt: Gissa faktorer

För att Python ska kunna skapa slumptal måste du använda dig av ett tillägg till språket. I Python kallas sådana tillägg för moduler.

För att använda en modul skriver du import följt av modulens namn, exempelvis

```
import random
```

som låter oss använda oss av slumptalsmodulen random.

Därefter kan du skapa olika slumptal genom flera olika funktioner. Kom ihåg att funktioner är programkod som någon annan person redan skrivit åt oss. Slumptalsfunktionerna ligger i modulen random och är efter import random färdiga att användas i ditt program.

Om vi vill skapa ett slumpvärde i form av ett heltal mellan 1 och 50 skriver du

```
random.randint(1, 50)
```

Du kan också stoppa in slumptalet i en variabel

```
slumptal = random.randint(1, 50)
```

Du ska skriva ett program där användaren ska få se ett slumptal från 1 till 100.

Användaren ska därefter ange två faktorer så att produkten av de två blir lika med slumptalet.

Programmet ska fortsätta till användaren har gissat rätt 10 gånger (för 10 oika produkter).

När programmet slutar ska det skriva ut hur många gånger användaren gissade fel.

Exempel:

Nämn två faktorer till 46.

2

23

*Rätt, för 2 * 23 = 46. Du har 1 rätt*

Nämn två faktorer till 69.

...

4.4. Projekt: Problemlösning med råstyrka

Eftersom datorer är snabba behöver vi inte alltid skriva så "smarta" program.

Vi kan i stället utnyttja snabbheten för att skapa en mängd möjliga lösningar och sedan pröva om de är korrekta.

För att lista alla jämna tal ska vi nu skapa alla tal och därefter undersöka om de är jämna.

Det gör vi genom att dela dem med två och se om svaret blir ett heltal. Om vi dividerar ett udda tal med två kommer vi däremot alltid att få ett decimaltal som svar (det kommer att sluta på ,5).

För att undersöka om svaret blir ett heltal jämför vi svaret med och utan decimaler. Om dessa två är lika är det ett heltal.

> *Exempel:*
>
> *Vi undersöker talet 7. 7 / 2 = 3,5. Eftersom 3,5 ≠ 3 är 7 inte ett jämnt tal.*
>
> *Vi undersöker talet 46. 46 / 2 = 23. Eftersom 23 = 23 är 46 ett jämnt tal.*

Nu ska vi undersöka vilka tal från 1 till 20 som är jämna.

```
for tal in range(1, 20):
    kvot = tal / 2
    if kvot == int(kvot):
        print (tal)
```

Att lösa problem på detta sätt kallas för "brute force", råstyrka.

1. Skriv ett program som hittar alla udda tal från 1 till 100 med hjälp av råstyrkemetoden.

2. Skriv ett program som hittar alla tal delbara med 3 från 1 till 100 med hjälp av råstyrkemetoden.

4.5. Projekt: Faktorisering

Alla tal kan delas upp som en produkt av två eller flera tal.

Exempel:

12 = 4 · 3

63 = 7 · 9 = 7 · 3 · 3

Nu ska vi studera hur vi kan dela upp tal som en produkt av två tal med hjälp av råstyrkemetoden. Det får vi genom att skapa alla möjliga kombinationer av två tal och sedan undersöka om produkten av de två är lika med det tal vi undersöker. (I det här fallet betyder ”alla möjliga kombinationer” produkten av alla tal från 1 till 100).

```
tal = 56
for faktor1 in range(1, 100):
    for faktor2 in range (1, 100):
        if tal == faktor1 * faktor2:
            print (faktor1, faktor2)
```

Här använder vi oss av två "nästlade" for-satser. Det fungerar så här:

- Först kommer faktor1 att vara 1.
- Därefter kommer faktor2 att i tur och ordning anta alla värden från 1 till 100. För varje faktor2 så undersöker vi om produkten av faktor1 (till en början 1) och faktor2 är lika med talet vi undersöker.
- Sedan får faktor1 värdet 2 och faktor2 kommer i tur och ordning åter att anta alla värden från 1 till 100.
- Detta fortsätter tills faktor1 har antagit alla värden från 1 till 100.

Prova metoden med några olika värden på tal.

Fundera på: Hur stora tal kan du faktorisera med metoden? Vad ska du ändra i programmet för att kunna faktorisera ännu större tal?

Denna metod kommer dock att skapa dubletter, det vill säga den kommer att hitta exempelvis både 2 · 28 och 28 · 2.

I vår inre for-sats behöver vi bara undersöka faktorer som går från faktor1 och uppåt.

Fundera på: Varför då?

```
tal = 56
for faktor1 in range(1, 100):
    for faktor2 in range (faktor1, 100):
        if tal == faktor1 * faktor2:
            print (faktor1, faktor2)
```

Vi kan dessutom göra vår metod ännu effektivare genom att inte undersöka större faktorer än talet självt (kom ihåg att range() bara skapar tal upp till ett mindre än det andra argumentet - därför måste vi lägga till 1).

```
tal = 56
for faktor1 in range(1, tal + 1):
    for faktor2 in range (faktor1, tal + 1):
        if tal == faktor1 * faktor2:
            print (faktor1, faktor2)
```

1. Hur stora tal kan du faktorisera med denna metod utan att det tar för lång tid?
2. Har du några ideer på hur programmet skulle kunna undvika att pröva alla tal? Du behöver inte skriva någon kod, bara beskriva något eller några sätt!
3. Skriv om testet (if-satsen) så att det använder operatorn %.
4. Skriv om funktionen så att den letar efter faktorisering med tre faktorer.

4.6. Projekt: Funktioner

(Innan detta projekt bör du har gjort Projekt: Problemlösning med råstyrka och Projekt: Faktorisering.)

Säg att vi vill hitta alla jämna till upp till något annat tal, exempelvis 40, 200 eller 3 000 000?

Det är både onödigt jobbigt, och dessutom en källa till fel, att gång på gång skriva ny kod som skapar alla dessa tal.

Vi kommer nu därför att övergå till ett annat sätt att arbeta med tal, funktioner.

```
def jamnaTal(uppTill):
    for i in range(1, uppTill):
        if (i / 2) == int(i / 2):
            print(i)
```

Du känner igen de tre rader med kod som med hjälp av råstyrke-metoden skapar en mängd tal och därefter undersöker vilka av dessa tal som är jämna.

På första raden har vi skrivit

```
def jamnaTal(uppTill):
```

På detta sätt skapar vi en funktion (”def” står för ”define”, definiera). Funktionen kommer att heta jamnaTal (”jämna tal”). Den har dessutom ett så kallat argument som vi döpt till uppTill. Ett argument är en variabel som låter oss anpassa funktionens programkod till olika värden.

Vi använder den på följande sätt:

```
jamnaTal(20)
```

Att skriva jamnaTal(x) där x har ett värde kallas för att ”anropa” funktionen. Det innebär att uppTill får värdet 20 och att koden efter def jamnaTal(uppTill) kommer att köras.

1. Skapa en funktion kallad uddaTal som skriver ut alla udda tal från 1 till ett värde som skickas till funktionen.

2. Skriv en funktion som faktoriserar ett tal upp till en viss gräns. Faktoriseringen ska ske med två tal.

def faktorisering(tal):

4.7. Projekt: Funktioner som ger svar

En funktion kan om den vill alltid skicka tillbaka ett värde. Det gör vi genom att skriva return följt av värdet den ska skicka tillbaka.

Ett exempel är en funktion som räknar ut kvadraten av ett tal:

```
def kvadrera(tal):
    svar = tal * tal
    return svar
```

Hur använder vi en funktion? Eftersom en funktion skickar tillbaka ett värde så kan vi antingen spara detta värde:

```
kvadrat = kvadrera(4)
print("Talet 4 har en kvadrat som är", kvadrat)
```

eller skriva ut det direkt:

```
print("Talet 4 har en kvadrat som är", kvadrera(4))
```

Observera att funktionen alltid måste skapas innan du kan använda den. Det innebär att den måste ligga innan du använder den i programmet.

1. Skriv en funktion som räknar ut och skickar tillbaka värdet på tal1 upphöjt till tal2.

2. Skriv en funktion som räknar ut summan från 1 till argumentet och skickar tillbaka det. Exempel: Du skickar in 4. Svaret ska då bli 10 eftersom 1 + 2 + 3 + 4 = 10.

3. Det finns ett matematiskt begrepp som heter fakultet. Det skrivs exempelvis 4! vilket betyder att du ska multiplicera alla tal från 1 upp till det angivna talet. $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$. Skriv en funktion som räknar ut fakulteten till argumentet.

4.8. Projekt: Primtal

Nu ska vi skriva en funktion som undersöker om ett tal är ett primtal!

Definition: Ett primtal är endast delbart med 1 eller sig själv. De första primtalen är 2, 3, 5, 7, 11.

Detta kan vi göra genom att

1. Gå igenom alla tal från 2 upp till ett mindre än talet själv. Kom ihåg att range() inte tar med ”till-talet”, därför blir det range(2, tal).
2. Dividera vårt tal med alla dessa tal i tur och ordning och se om divisionen går jämt ut.

Använd råstyrkemetoden!

```
def primtal(tal):
    for i in range(2, tal):
        if tal % i == 0:
            return False
    return True
```

Den här funktionen skriver inte ut något själv. I stället skickar den tillbaka ett värde som talar om om talet är ett primtal i form av antingen False (falskt) eller True (sant).

Vår primtalsfunktion kan vi använda på följande sätt:

```
mitt_tal = 47
ar_tal_primtal = primtal(mitt_tal)
if ar_tal_primtal:
    print(mitt_tal, "är ett primtal")
else:
    print(mitt_tal, "är inte ett primtal")
```

1. Hur stora tal kan du undersöka utan att det tar för lång tid?

2. Kan du föreslå något effektivare sätt att undersöka om ett tal är ett primtal? Är det alltid nödvändigt att undersöka alla tal upp till talet - 1?

4.9. Projekt: Slumptal

För att få ditt program att fungera olika varje gång du kör det kan du använda dig av slumptal.

För att Python ska kunna skapa slumptal måste du använda dig av ett tillägg till språket. I Python kallas sådana tillägg för moduler och det finns en mängd olika moduler för olika saker, exempelvis statistik, grafik och så vidare.

För att använda en modul skriver du import följt av modulens namn, exempelvis

```
import random
```

som låter oss använda oss av slumptalsmodulen random.

Därefter kan du skapa olika slumptal genom flera olika funktioner. Kom ihåg att funktioner är programkod som någon annan person redan skrivit åt oss. Slumptalsfunktionerna ligger i modulen random och är efter import random färdiga att användas i ditt program.

Om vi vill skapa ett slumpvärde i form av ett heltal mellan 1 och 100 skriver du

```
random.randint(1,100)
```

Du kan också spara slumptalet i en variabel

```
slumptal = random.randint(1,100)
```

Ett program för att skriva ut 10 slumptal kan se ut så här:

```
import random
raknare = 0
while (raknare < 10):
    slumptal = random.randint(1,100)
    print(slumptal)
    raknare = raknare + 1
```

Skapa ett program som låter användaren gissa ett tal mellan 1 och 100. Programmet ska för varje gissning skriva ut om svaret var för lågt, för högt eller rätt. Om användaren gissar rätt ska programmet skriva ut hur många gånger användaren var tvungen att gissa.

Använd dig av while för att skapa en slinga. Slingan ska fortsätta tills gissningen är lika med slumptalet.

Du bör också använda dig av en variabel som räknar gissningarna och som ökar med ett varje gång användaren gissar.

4.10. Projekt: Talbaser

Vi arbetar i större delen av världen med talbasen 10, det vill säga varje position som en siffra står ”till vänster” ökar dess värde med 10.

$$42 = 4 \cdot 10 + 2$$

$$4711 = 4 \cdot 1000 + 7 \cdot 100 + 11 \cdot 10 + 1 \cdot 1$$

eller om du känner till tio-potenser

$$4711 = 4 \cdot 10^3 + 7 \cdot 10^2 + 11 \cdot 10^1 + 1 \cdot 10^0$$

Vi kan räkna med andra talbaser också.

1. Skriv ett program som tar ett tal beskrivet i en annan angiven talbas och skriver ut dess värde med vår talbas.

2. Skriv ett program som tar ett tal angivet i talbasen 10 och skriver ut det i en annan angiven talbas.

5. Område: Algebra

5.1. Projekt: Formler

Om vi har en formel som vi ofta vill använda så är det enkelt att skapa den i Python i form av en funktion. Ett exempel är den fysikaliska formeln:

$$\text{sträckan} = \text{hastigheten} \cdot \text{tiden},\ s = v \cdot t.$$

```
def s(v, t):
    return v * t
```

Vi använder den för att exempelvis ta reda på hur många kilometer en bil som åker i 70 km/h kommer på 8 timmar.

```
print("Bilen åker", s(70, 8), "km.")
```

1. Skriv ett program där användaren kan mata in hastigheten och tiden. Programmet ska därefter skriva ut sträckan.

2. Skriv en funktion som räknar ut hastigheten om du vet tiden och sträckan. Skriv ett program liknande det i uppgift 1.

3. Funktioner är också bra för enhetsomvandling. Skriv en funktion som omvandlar kilometer i timmen (km/h) till meter per sekund (m/s).

5.2. Projekt: Talföljder

En talföljd är en uppräkning av tal enligt någon särskild metod eller mönster.

Exempel: 2, 4, 6, 8, ... är talföljden bestående av alla jämna tal.

En del talföljder kan vi beskriva med hjälp av algoritmer, alltså sätt att räkna ut dem.

Vår jämna talföljd har vi tidigare beskrivit (upp till 100):

```
tal = 0
while tal < 100:
    tal = tal + 2
    print(tal)
```

Alla jämna tal är en aritmetisk talföljd eftersom differensen mellan två tal alltid är konstant.

Om kvoten mellan två tal i serien är konstant kallas talföljden för geometrisk. Det innebär att du räknar ut nästföljande tal i serien genom att multiplicera med en konstant. Exempel:

```
    tal = tal * 2
```

1. Skriv en funktion som skriver ut en valfri aritmetisk talföljd upp till det tal som skickas som argument till funktionen.

2. Skriv en funktion som skriver ut en valfri geometrisk talföljd upp till det tal som skickas som argument till funktionen.

5.3. Projekt: Prövning av ekvationer

Det är relativt enkelt att göra beräkningar med siffror i Python. Det är inte lika lätt att arbeta med symboler som x och y. (Det finns dock ett tillägg, bibliotek, för detta kallat SymPy.)

Men Python kan snabba upp prövningen av dina ekvationslösningar!

Säg att vi har en ekvation: 13x - 9 = 30.

Du kan då pröva din lösning på x genom att skapa en funktion för vänsterledet (VL).

```
def vl(x):
    return 13 * x - 9
```

Du kan nu pröva en lösning genom att använda dig av funktionen:

```
vl(4)
```

och jämföra svaret med högerledet (HL).

Eller så gör du det direkt genom:

```
vl(4) == 30
```

Då kommer Python att skriva ut om det stämmer (”True”) eller inte stämmer (”False”).

1. Leta upp en ekvation som du löst där HL är ett tal (en konstant) och VL innehåller x (eller någon annan variabel). Skriv ett program där användaren får mata in två tal, x samt talet i HL. Lägg in VL i en funktion. Programmet ska skriva ut om VL = HL.

5.4. Projekt: Fibonacci

(Innan detta projekt bör du har gjort Projekt: Problemlösning med råstyrka.)

Det finns många spännande talföljder. En av de mest kände är fibonacci-talen. Där räknas varje tal ut som summan av de två föregående talen:

0, 1, 1, 2, 3, 5, 8, 13, ...

Så här kan vi skriva ut alla fibonacci-tal upp till 1000.

```
# Fib A
a, b = 0, 1
while b < 1000:
    print(a, b, a + b)
    a, b = b, a+b
```

På den första raden använder vi Pythons möjlighet att tilldela värden till två variabler på samma rad. Variabeln a kommer att få värdet 0. Variabeln b kommer att få värdet 1. Variabeln b kommer hela tiden att vara nästa värde i talserien, medan a kommer att vara det föregående.

Det kan vi göra på ett smart sätt i slingen genom att även där använda dubbel tilldelning. Variabeln a kommer att få värdet i b, medan b kommer att bli lika med summan av (det gamla) värdet i a adderat med det gamla värdet i b.

Om vi skriver programmet så här:

```
# Fib B
a, b = 0, 1
while b < 1000:
    print(a, b, a + b)
    a = b
    b = a+b
```

kommer programmet inte att fungera. Varför inte? Prova!

Vi kan skriva ett fungerande program så här:

```
# Fib C
a, b = 0, 1
while b < 1000:
    print(a, b, a + b)
    gamla_a = a
    a = b
    b = gamla_a + b
```

Vi sparar värdet på a så att vi kan tilldela b värdet av gamla a adderat med gamla b.

1. Förklara varför programmet Fib B inte fungerar som tänkt.

2. Skriv om algoritmen som en funktion kallad fibonacci(a, b).

3. Skriv en funktion som med hjälp av råstyrke-metoden undersöker om ett tal ingår i fibonacci-talserien.

4. Sök på nätet efter några områden där fibonacci-tal förekommer eller används. Beskriv dem.

5.5. Projekt: Egen talföljd

Hitta på en egen talföljd och beskriv den med hjälp av en funktion.

Skriv också en funktion som tar reda på om ett tal ingår i denna talföljd.

Exempel:

```
def minTalfoljd():
    # kod som skriver ut talföljden
def ingarIMinTalfoljd(tal):
    if # test för att se om värdet på variabeln tal ingår i talföljden
        return True
    else:
        return False
```

5.6. Projekt: Euklides algoritm

Största gemensamma delare (SGD) innebär det största tal som delar två eller flera tal.

> *Exempel:*
>
> *sgd(15, 6) = 3 eftersom 15 / 3 = 5 och 6 / 3 = 2.*

Vi kan använda sgd för två tal om vi vill skriva om ett bråk i dess enklaste form.

> *Vi ska skriva om 15/6 i dess enklaste form. Eftersom sgd(15, 6) = 3 så delar vi både 15 och 6 med 3 och får då 5/2.*

Vi har tidigare tittat på hur vi kan hitta en gemensam nämnare genom att skapa en stor mängd tal och sedan undersöka vilka två som skapar det eftersökta talet.

Det finns en berömd algoritm som är smartare. Den kallas för Euklides algoritm. Kom ihåg att en algoritm och ett program är väldigt nära släkt. Vi borde därför utan problem kunna översätta algoritmen till programkod.

Euklides algoritm säger följande: SGD för två tal förändras inte om det största talet ersätts med differensen med det mindre talet.

> *sgd(15, 6) = sgd(9, 6) = sgd(3, 6) = sgd(3, 3)*

Fundera på: Varför är det så?

Algoritmen gör att talen hela tiden minskar i storlek och den fortsätter tills de två talen är lika.

```
def sgd(x, y):
    while x != y:
        if (x > y):
            x = x - y
        else:
            y = y - x
    return x

print(sgd(15, 6))
```

1. Lägg till utskrifter så att du förstår och kan förklara vad som händer i algoritmen.

2. Sök på nätet och se om du kan hitta ännu kortare lösningar till sgd() (kallas för gcd - greatest common divisor - på engelska).

3. (Svårare.) Versionen av Euklides algoritm som vi visade ovan kan ta många steg på sig för stora tal. Vi kan istället ersätta det större talet med resten om vi dividerar det med det mindre talet. Vi ska då stanna om vi får en rest som är lika med 0.

Skriv om algoritmen så att den arbetar på detta smartare sätt.

5.7. Projekt: Intervallhalvering

Här är ett enkelt program för att låta användaren gissa ett tal mellan 1 och 100:

```
import random
tal = random.randint(1,100)
gissning = 0
while gissning != tal:
    gissning = int(input("Gissa ett tal:"))
    if gissning > tal:
        print("Du gissade för högt!")
    if gissning < tal:
        print("Du gissade för lågt!")
print("Grattis! Du gissade rätt!")
```

Det går att gissa talet genom att helt enkelt pröva en massa olika tal. Men kan du komma på en bra strategi för att hitta talet?

Ett effektivt sätt att gissa ett slumpmässigt val tal är intervallhalvering. Metoden går ut på följande:

> *Talet vi ska gissa är mellan 1 och 100.*
>
> *Vi gissar då först på ett tal som ligger mitt emellan, 50.*
>
> *Om vi får reda på att det är för stort så gissar vi på ett tal som ligger mitt mellan 1 och 50, det vill säga 25.*
>
> *Om det är för litet så gissar vi på ett tal som ligger mitt mellan 25 och 50, det vill säga 75.*
>
> *Och så vidare...*

1. Kör programmet ovan och pröva att gissa talet genom intervallhalveringsmetoden. Anteckna hur många gånger du måste gissa. Hur många gånger måste du som högst gissa?

2. Ändra i programmet så att du provar några andra intervall, exempelvis 1-25, 1-40, 1-1000. Kan du hitta något samband mellan intervallets storlek och högsta antalet gissningar?

Nu har vi en algoritm för att gissa tal som vi kan överföra till ett program. Men denna gång låter vi datorn gissa ett tal som du tänker på.

```
# a och b anger start och slut på intervallet
a = 1
b = 100
```

```
fraga = "Tänk på ett tal mellan " + str(a) + " och " + str(b)
tal = int(input(fraga))
resultat = ""
while resultat != "R":
    # Vi gissar på talet mitt i intervallet
    gissning = int((a+b)/2)
    print("Datorn gissar", gissning, ".")
    resultat = input("För högt (H), för lågt (L) eller rätt (R)?")
    if resultat == "H":
        b = gissning - 1
    if resultat == "L":
        a = gissning + 1
print("Grattis datorn!")
```

1. Inför en räknare så att programmet i slutet skriver ut hur många gånger datorn gissade.

2. Se efter hur många gånger du kan få datorn att gissa för samma intervall som du själv använde när du gissade.

5.8. Projekt: Ekvationslösning med intervallhalvering 1

Säg att vi har en ekvation som vi har problem med att lösa (kanske har vi inte ens lärt oss att räkna ut den).

Vi kan alltid skriva om den på formen:

ekvation = 0

genom att ta allt som står i HL och dra ifrån det i VL.

Exempel

x · x - x + 4 = 6

blir

x · x - x + 4 - 6 = 0

vilket blir

x · x - x - 2 = 0

Det betyder att det finns ett x-värde där den här ekvationen blir 0 (om det finns någon lösning vill säga). Vi ska alltså hitta ekvationens nollpunkt, det vill säga när VL blir 0.

Kan du hitta lösningen genom att pröva några olika värden på x?

Vi ska nu låta datorn gissa och använda intervallhalveringsmetoden för att göra gissningen effektiv.

Programmet kan bara hitta värdet på x om intervallet som den gissar i täcker denna 0-punkt. Vi gissar i detta faller i intervallet -1000 till 1000.

Fundera på: Varför?

För att denna algoritm ska fungera måste variabeln svar börja med ett tal skiljt från 0.

Fundera på: Varför?

```
a = -1000
b = 1000
svar = 1

# Denna ekvation har lösningen x = 2
def f(x):
    return x * x - x - 2

while svar != 0:
    gissning = int((a + b) / 2)
    svar = f(gissning)
```

```
    if svar < 0:
        a = gissning
    if svar > 0:
        b = gissning
print("Lösning", gissning)
```

I detta program hittar du en rad som börjar med en så kallad ”brädgård”, eller ”hash-tecken”. Det innebär att raden är en kommentar och att Python kommer att ignorera allt som står på raden efter #.

1. Prova programmet. Vilka begränsingar ser du i detta sätt att lösa ekvationer?

5.9. Projekt: Ekvationslösning med intervallhalvering 2

Vi har bara tittat på ekvationer som har heltal som lösningar. Vi kan få denna lösningmetod att hitta närmevärden, alltså lösningar som är nästan rätt, även till ekvationer som inte har heltal som lösningar.

```
# Vi gissar inom intervallet -1000 - 1000
a = -1000
b = 1000
svar = 1 # ett tal != 0 som vi börjar med

# Denna ekvation har lösningen 1.5
def f(x):
    return x * 4 - x - 4.5

while abs(a - b) > 0.1:
    gissning = (a + b) / 2
    print(a, b, gissning)
    svar = f(gissning)
    if svar < 0:
        a = gissning
    if svar > 0:
        b = gissning
print("Lösning", gissning)
```

Funktionen abs() skickar tillbaka absolutvärdet av ett uttryck, det vill säga ”tar bort” alla minus-tecken.

abs(1) = 1

abs(-1) = 1

abs(-10000) = 10000

Raden while abs(a - b) < 0.1 undersöker intervallets storlek. Om det är mindre än 0,1 så är vi nöjda med lösningen. Funktionen abs() ser till att intervallet, a - b, alltid är positivt.

1. Pröva med andra ekvationer som du letar upp.
2. Pröva att ändra 0.1 till 0.01 eller ännu mindre. När börjar det ta för lång tid?

5.10. Projekt: Kryptografi

Krypto, även kallade chiffer, används för att skicka meddelanden som är hemliga och som inte ska kunna läsas av någon annan än mottagaren.

Ett av de enklaste kryptona är ersättningskryptot som också kallas för Caesarrullning eftersom Julius Caesar använde det när han kommunicerade med sin landsman Cisero.

Det går ut på att förskjuta bokstäverna åt något håll så att exempelvis A blir B, B blir C, Ö blir A och så vidare.

I Python kan vi använda for för att gå igenom alla tecken i en sträng.

```
klartext = "MATEMATIK"
kryptotext = ""
for bokstav in klartext:
    bokstav_nummer = ord(bokstav)
    bokstav_nytt_nummer = bokstav_nummer + 1
    krypto_bokstav = chr(bokstav_nytt_nummer)
    kryptotext = kryptotext + krypto_bokstav
```

För att förstå hur det här programmet fungerar behöver vi veta hur Python (och datorer i allmänhet) sparar text. Varje bokstav har ett nummer. Av tradition har ”A” värdet 65, ”B” värdet 66 upp till ”Z” som har värdet 90.

De svenska tecknena har andra värden som du kan se om du skriver exempelvis

```
>>> print(ord("Å"))
197
```

För enkelhetens skull kommer vi därför bara arbeta med det engelska alfabetet från A till Z eftersom de alla kommer i ordning.

Här använder vi två inbyggda funktioner i Python. ord() returnerar bokstavens ordningsnummer och chr() tar ett ordningsnummer och returnerar bokstaven.

ord("B") = 66

chr(66) = "B"

1. Programmet kommer inte att förvandla bokstaven Z till A. Modifiera programmet så att Z hanteras korrekt. Slå på nätet upp hur ”mod” fungerar i Python.

2. Skriv ett program där användaren får mata in ett tal och sedan en sträng i klartext. Programmet ska förskjuta texten i alfabetet med så många steg som anges av variabeln tal. Därefter ska det skriva ut den krypterade strängen.

3. Skriv ett program som tar en krypterad text och antalet förskjutningar och återställer den till klartext. Eller kan du kanske använda programmet du skrev för att

kryptera texten även till detta?

4. (Svårare.) Tänk dig att du får en krypterad text där du inte vet hur många steg bokstäverna förskjutits. Hur kan du lösa ett sådant krypto?

5.11. Projekt: Komplett kryptering

Genom exempelvis

```
>>> print(ord("Å"))
197
```

kan du ta reda på ordningsnummer för de svenska tecknena åääöÅÄÖ.

Du kan även ta reda på ordningsnummer för alla gemener.

> Modifiera krypteringsalgoritmen i Projekt: Kryptografi så att den kan hantera både gemener och svenska tecken. Lägg hela algoritmen i en funktion.

6. Område: Geometri

6.1. Projekt: Månghörning

(Innan du gör detta projekt bör du ha gjort Python: Lektioner: Turtle.)

Skriv en funktion som ritar en månghörning.

```
def ritaManghorning(horn)
```

Hur många hörn måste du ange för att månghörningen ska likna en cirkel?

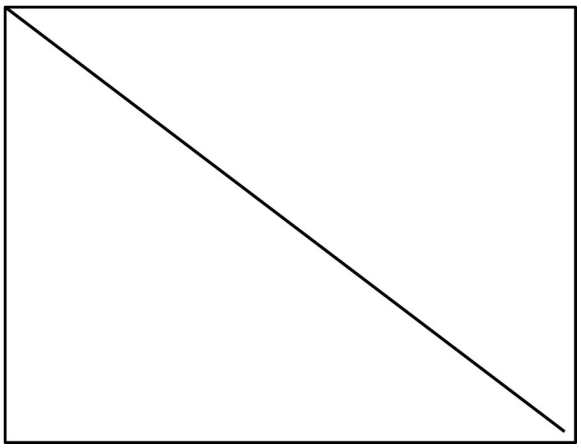
Lägg till kod som låter användaren ange hur många hörn figuren ska ha och sedan ritar den.

Beskriv hur din kod fungerar.

6.2. Projekt: Vinkelsumma

En triangel har vinkelsumman 180 grader.

En kvadrat har vinkelsumman 360 grader. Detta går att visa genom att dela in kvadraten i två trianglar.



Det går att dela in alla månghörningar i trianglar och på detta sätt räkna ut deras vinkelsumma.

1. Skriv ett program där användaren får mata in antal hörn i månghörningen.

Programmet ska sedan rita ut månghörningen.

Det ska därefter räkna ut vinkelsumman genom att använda sig av kunskapen att varje triangel har vinkelsumman 180 grader.

Beskriv hur din kod fungerar.

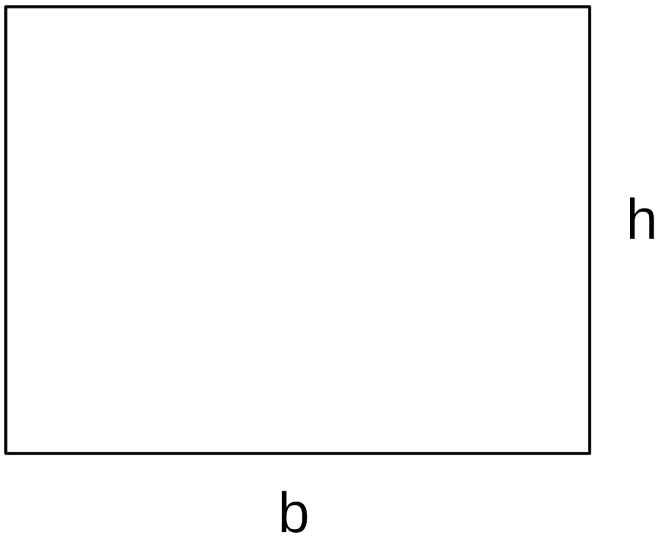
2. Modifiera ditt program så att månghörningen delas in i trianglar.

6.3. Projekt: Rektangelns omkrets och area

En rektangel har en omkrets som är lika med summan av dess sidor.

Arean ges av formeln

$$A = b \cdot h$$



1. Skriv ett program som går igenom alla värden på bredd från 10 till 90, där höjd är 100 - bredd. Omkretsen kommer alltså alltid att vara (bredd + höjd) · 2 = 200.

För varje kombination av bredd och höjd ska programmet rita ut kvadraten och räkna ut dess area. Areans värde ska skrivas ut.

2. Modifiera programmet så att det håller reda på den största arean och skriver ut denna innan programmet avslutas.

Vilka slutsatser kan du dra om rektangelns area och dess bredd och höjd?

Att visa på samband är inte lika mycket värt som ett bevis. Men att visa samband kan ändå peka ut en riktning där vi kan gräva djupare.

6.4. Projekt: Digitalkamera

1. Ta reda på hur många pixlar din mobil fotograferar med. Hur många pixlar är det?

Minnet i en dator räknas vanligen i bytes, där en byte kan spara ett värde mellan 0 till 255.

I en dator (och i din mobil) lagras varje pixel vanligen som tre tal mellan 0 och 255. Det gör att varje pixel kräver 3 bytes.

2. Skriv ett program där användaren anger bredd och höjd på bilden i antal pixlar. Programmet ska sedan räkna ut hur stor bilden är räknat i bytes.

3. Ta reda på hur bilder lagras genom att söka på nätet.

I praktiken komprimeras bilder och sparas som en så kallad JPG-fil. Det gör att bilden inte tar så mycket plats men å andra sidan försvinner en del information (som vanligtvis inte kan uppfattas av ögat).

6.5. Projekt: Bærtling

Olle Bærtling är namnet på en svensk konstnär som levde 1911-1981. Han arbetade mycket med geometriska former. Sök gärna upp några av hans verk på nätet!

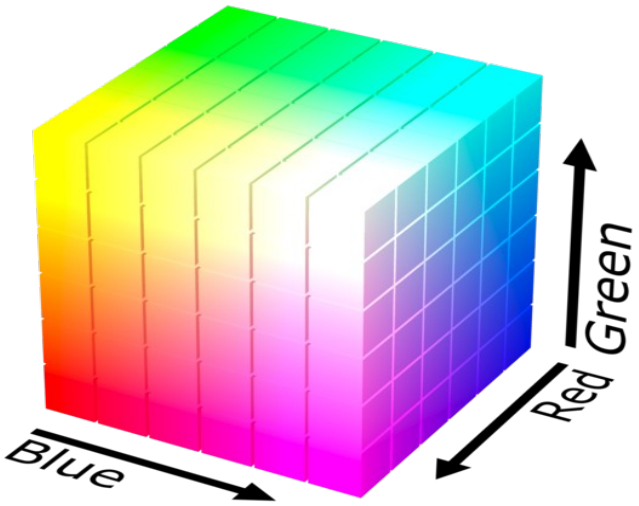
För att kunna göra egna geometriska bilder får du här tips på några ytterligare möjligheter som Turtle erbjuder.

Genom funktionen

```
    padda.pencolor(r, g, b)
```

kan du ändra den färg som Turtle ritar med.

De tre värdena r, g och b står för rött, grönt och blått och anges som var sitt tal från 0 till 255. De kan blandas och kan på så sätt skapa olika värden. Tänk på dem som tre färgade strålkastare.



(Källa: https://commons.wikimedia.org/wiki/File:RGB_color_solid_cube.png)

Pythons Turtle-modul kan också ha färgvärden i intervallet 0 - 1.0. För att Python alltid ska arbeta med värden från 0 - 255 så måste vårt program ha följande två rader i början:

```
fonster = turtle.Screen()
fonster.colormode(255)
```

De figurer du skapar kan sedan fyllas med valfri färg.

```
padda.begin_fill()
padda.fillcolor(255, 127, 0)
padda.forward(100)
padda.left(120)
padda.forward(100)
```

```
padda.left(120)
padda.forward(100)
padda.left(120)
padda.end_fill()
```

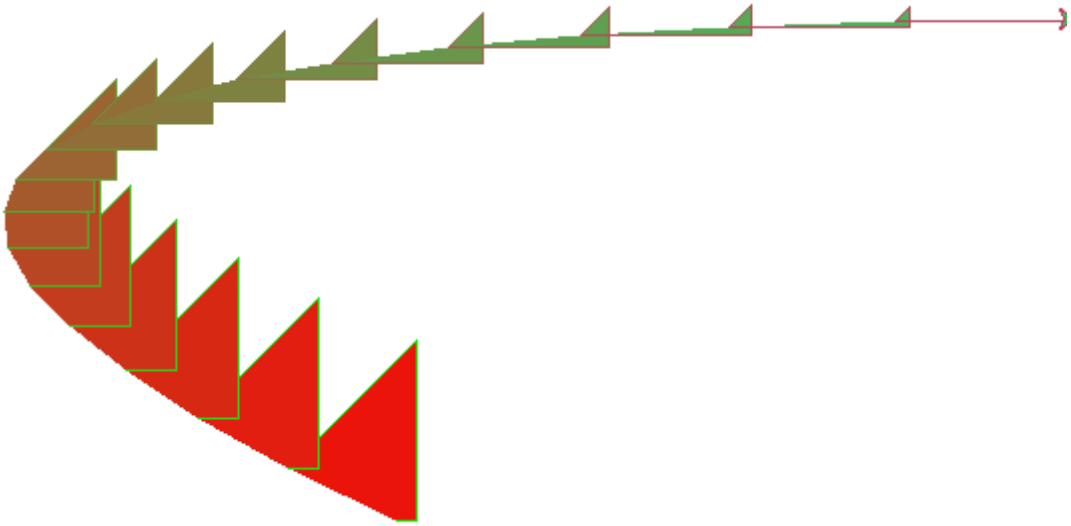
Vill du att Turtle ska rita snabbare?

```
padda = turtle.Turtle()
padda.speed("fastest")
```

Med hjälp av slingor kan du skapa figurer som upprepas. Låt dem gärna gå oväntade vägar och svänga så att de inte skapar hela figurer. Här är ett exempel som använder de nya funktionerna.

```
import turtle

fonster = turtle.Screen()
fonster.colormode(255)

padda = turtle.Turtle()
padda.speed("fastest")

def ritaFigur(bredd, hojd):
    padda.forward(bredd)
    padda.left(90)
    padda.forward(hojd)
    padda.left(135)
    padda.forward(hojd)
    padda.left(135)

for b in range(10, 100, 5):
    h = 100 - b

    # varje färg ska räknas ut som ett värde
    # som hamnar mellan 0 till 255
    farg_r = 2 * b
    farg_g = 2 * b
    farg_b = b

    padda.begin_fill()

    padda.pencolor(farg_r, 255 - farg_g, farg_b)
    padda.fillcolor(255 - farg_r, farg_g, farg_b)
```

```
    ritaFigur(b, h)
    padda.end_fill()
```



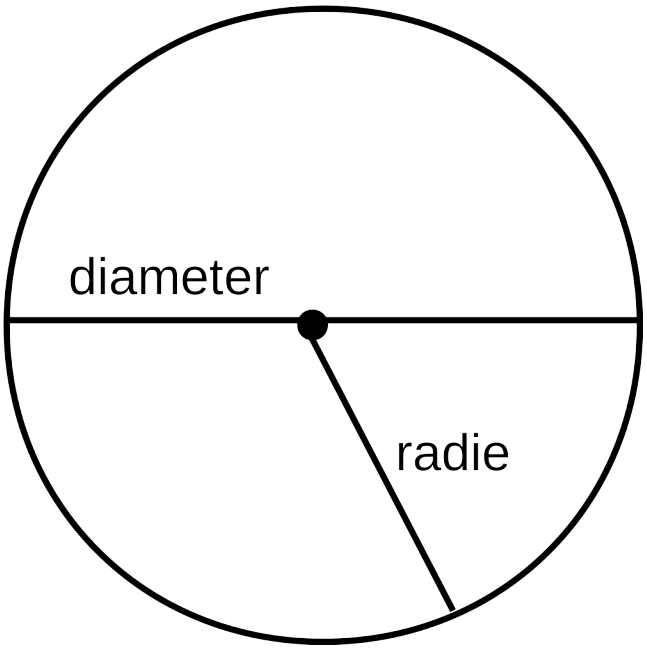
Observera hur vi har delat upp programmet i en funktion som ritar figuren och en som upprepar ritandet och anger färg. Vad är fördelen med denna uppdelning?

Skapa nu din egen konst! Inspireras av exemplet men skapa gärna något eget!

Beskriv hur din kod fungerar.

6.6. Projekt: π

π är en konstant som bland annat anger förhållandet mellan en cirkels omkrets och diameter.



π kan beräknas på många olika sätt. Eftersom π är irrationellt går talet inte att skriva ut exakt med siffror.

$$\pi \approx 3{,}14159\ 26535\ 89793$$

Här kommer några metoder som kan överföras till algoritmer och program.

Pröva minst två av dem och jämför hur snabbt de når ett korrekt värde med antal önskade decimaler.

Finns det andra för- och nackdelar med metoderna?

6.6.1. Gottfried Leibniz formel

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$$

Skriv ett program som räknar ut pi med Leibniz formel. Lägg beräkningen i en funktion som tar antalet bråk som indata.

6.6.2. John Wallis formel

$$\frac{\pi}{2} = \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7} \ldots$$

Skriv ett program som räknar ut pi med Wallis formel. Lägg beräkningen i en funktion som tar antalet tal i täljaren (eller nämnaren) som indata.

6.6.3. Eulers formel

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} \ldots$$

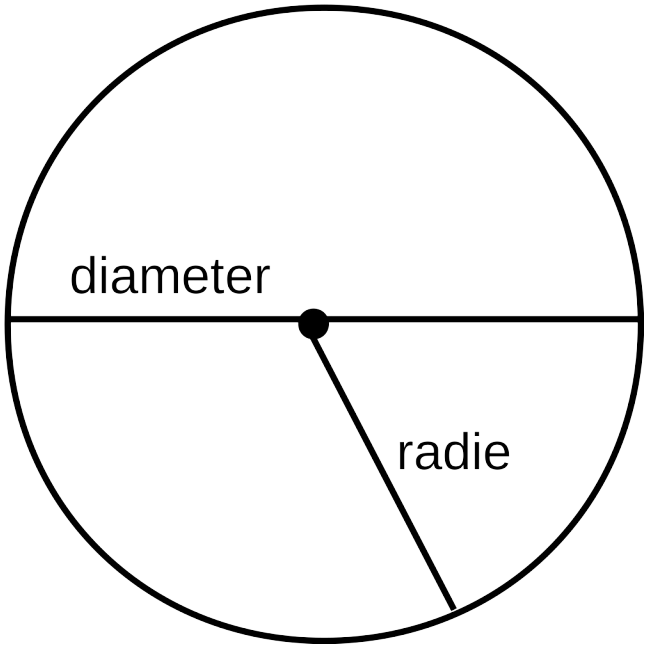
Skriv ett program som räknar ut pi med Eulers formel. Lägg beräkningen i en funktion som tar antalet bråk som indata.

6.7. Projekt: Cirkelns omkrets och area

En cirkels omkrets ges av formeln

$$O = \pi \cdot d$$

där O är omkretsen och d är diametern.



Du kan rita cirklar i Python med

```
import turtle
padda = turtle.Turtle()
padda.circle(50)
```

där 50 är radien på cirkeln.

Om du vill använda dig av π så kan du importera modulen math.

```
import math
print(math.pi)
```

Skriv ett program som

1. har en funktion som ritar ut en kvadrat med omkretsen x

2. har en funktion som ritar ut en cirkel med omkretsen x

Kan du få programmet att rita ut kvadraten och cirkeln med samma mittpunkt?

En cirkels area ges av formeln

$$A = \pi \cdot r^2$$

där A är arean och r är radien.

Vilka av de två figurerna (med samma omkrets) har den största arean? Bygg ut ditt program så att det räknar och skriver ut de två areorna.

Med en viss längd på ett staket, vilken av de två formerna (kvadrat och cirkel) inhägnar det största markområdet?

6.8. Projekt: Volym

Ta reda på vilken area och volym ett rätblock har.

Ta reda på vilken area och volym ett klot har.

Du har ett byggmaterial som kan täcka 10 m^2. Med vilken form (rätblock eller klot) kan du innesluta den största volymen?

6.9. Projekt: Skala

Skriv ett program som frågar användaren efter längdskalan och därefter räknar och skriver ut areaskalan och volymskalan.

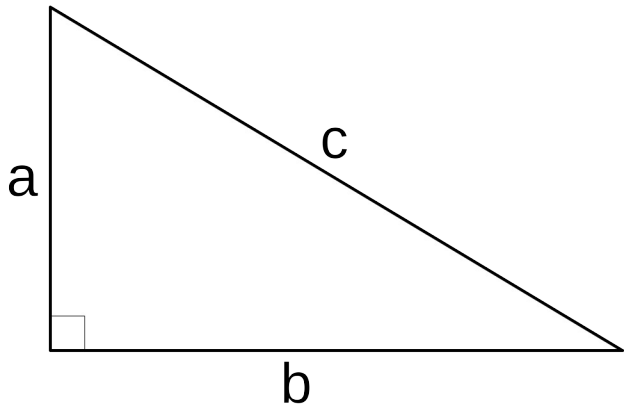
6.10. Projekt: Pythagoras sats

Pythagoras sats säger att

hypotenusan i kvadrat är lika med summan av sidorna i kvadrat

$$c^2 = a^2 + b^2$$

i en rätvinklig triangel.



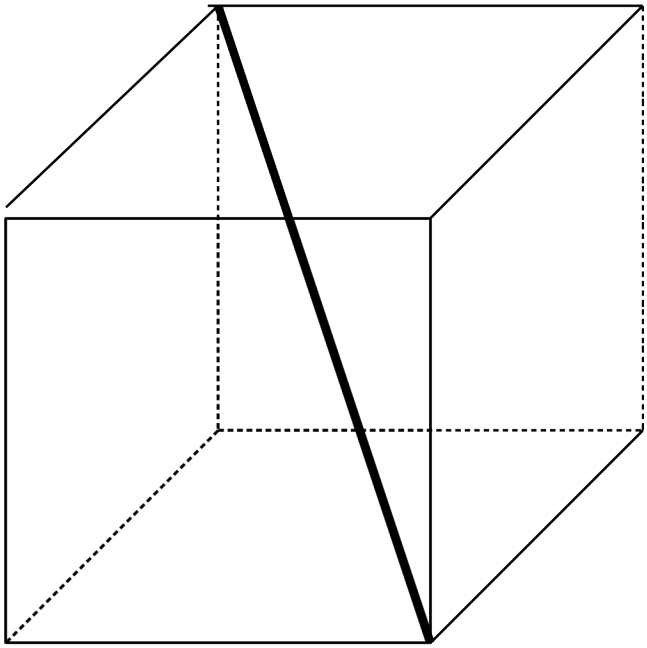
Men påståendet går även att vända på. Om Pythagoras sats inte stämmer så är triangeln inte rätvinklig.

Skriv ett program och en funktion

```
def triangelRatvinklig(a, b, c):
```

som returnerar True om triangeln är rätvinklig, det vill säga om Pythagoras sats stämmer, annars False.

6.11. Projekt: Rymddiagonal i kub



Skapa en funktion som räknar ut rymddiagonalen för en kub med sidan a.

Ett tips är att lägga in formeln för Pythagoras sats i en funktion.

7. Område: Samband och förändring

7.1. Projekt: Procent

Skriv ett program där användaren får ange delen och det hela. Programmet ska därefter skriva ut hur många procent delen är. Svaret ska skrivas både i decimal- och procentform.

7.2. Projekt: Lån

Skriv ett program där användaren får ange årsräntan på ett lån samt hur många år lånet är på.

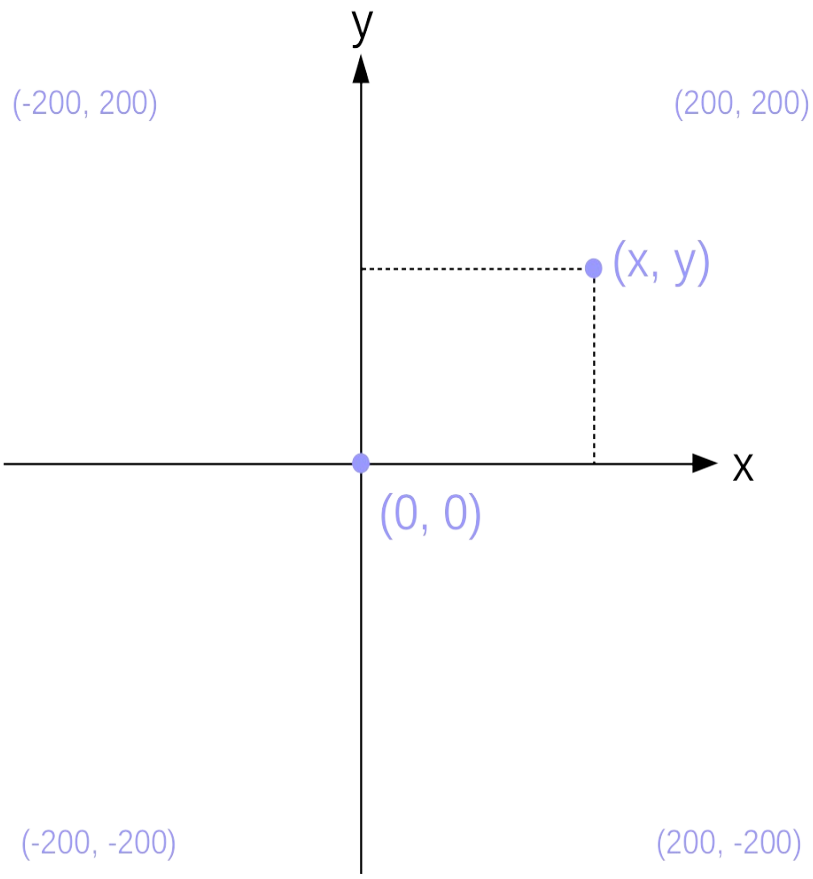
1. Anta att ingen amortering sker. Hur mycket växer lånet?

2. Modifiera programmet så att användaren kan ange en fast amortering (avbetalning) per år. Beräkna och skriv ut hur lånet förändras över åren tills det är betalt.

3. Modifiera programmet så att användaren kan ange en procentuell amortering (avbetalning) per år. Beräkna och skriv ut hur lånet förändras över åren tills det är betalt.

7.3. Projekt: Koordinatsystem

Turtle använder sig av följande koordinatsystem:



Vi kan rita ut x- och y-axlar:

```
import turtle
padda = turtle.Turtle()

def ritaKoordinatsystem(bredd, hojd):
    padda.penup()
    padda.goto(-bredd, 0)
    padda.pendown()
    padda.goto(bredd, 0)
    padda.penup()
    padda.goto(0, -hojd)
    padda.pendown()
    padda.goto(0, hojd)

ritaKoordinatsystem(200, 200)
```

Vi kan rita ut en punkt på koordinat x, y:

```
padda.penup()
padda.goto(x, y)
padda.pendown()
padda.dot(5) # 5 är storleken på punkten
```

vilket vi kan lägga i en funktion

```
def ritaPunkt(x, y):
    padda.penup()
    padda.goto(x, y)
    padda.pendown()
    padda.dot(5)
```

och använda så här

```
ritaPunkt(50, 50)
```

1. Skriv ut ett program som ritar ett koordinatsystem och sätter ut punkterna (45, 127), (3, 9), (-70, -190).

2. Skriv en funktion

   def ritaLinje(x1, y1, x2, y2)

som ritar en linje från punkt (x1, y1) till punkt (x2, y2).

3. Skriv om ditt program så att det kan hantera skala. Skala ligga i en variabel, exempelvis

   skala = 100

vilket motsvarar skala 1:100.

Programmet ska då kunna skriva ut exempelvis punkten (15000, 10000) så att den hamnar i koordinatsystemet (på skärmen).

7.4. Projekt: Linjediagram

Se till att du förstår hur listor fungerar genom att göra Python: Lektioner: Listor.

1. Skriv ett program som ritar ut ett koordinatsystem och ritar ut följande punkter som ska lagras i en lista: (-137, -137), (-50, 50), (137, 137), (50, -50).

2. Modifiera ditt program så att det även drar streck mellan punkterna. I vilken ordning måste punkterna ligga för att linjen ska bli korrekt?

Du har nu ett program som kan rita linjediagram.

3. Modifiera ditt program så att det efterfrågar x- och y-koordinater från användaren, lagrar dem i en lista och sedan ritar ut dem och drar en linje mellan dem.

Hur ska du hantera att punkterna kanske kommer i ”fel” ordning?

7.5. Projekt: Stolp- och stapeldiagram

(Se till att du gjort Projekt: Koordinatsystem och Projekt: Linjediagram.)

1. Skriv ett program som utifrån en lista med värden skriver ut ett stolpdiagram.
2. Skriv ett program som utifrån en lista med värden skriver ut ett stapeldiagram.

7.6. Projekt: Proportionalitet och linjära samband

Skriv ett program som ritar ut ett koordinatsystem och därefter ritar ut ett linjärt samband i detta.

Programmet ska exempelvis kunna rita ut kurvan

$y = 4 \cdot x$

7.7. Projekt: Kast av boll

En boll som kastas upp i luften kommer såklart att nå marken igen. Men dess rörelse antar formen av en så kallad kastparabel där höjden h är:

$$h = h_0 + v_{y0} \cdot t - \frac{g \cdot t^2}{2}$$

där h_0 är starthöjden, v_{y0} är hastigheten i höjdled, g är gravitationskonstanten och t är tiden. Vi bortser från luftmotståndet.

Läs mer om kastparabler: https://sv.wikipedia.org/wiki/Kastparabel

Om du tycker denna formel är svår att förstå så tänk bara på att bollens höjd måste bero på hur lång tid som har gått sedan du kastade den. Efter att en viss tid gått är höjden 0 och bollen slår i marken.

Ett exempel:

$$h = 1 + 5 \cdot t - \frac{10 \cdot t^2}{2}$$

1. Skriv om denna formel som en funktion i Python.

   def kastparabel(t):

2. Skriv ett program som ritar upp ett diagram och därefter ritar ut bollens höjd som en funktion av tiden.

Tips! Om du ritar ett diagram med storleken 200 x 200 så bör du dividera tiden med 100 och multiplicera höjden med 100.

Läs mer: https://www.matteboken.se/lektioner/skolar-9/uttryck-ekvationer-och-funktioner/koordinatsystem-och-grafer

7.8. Projekt: Räta linjens ekvation

Räta linjens ekvation:

$$y = k \cdot x + m$$

där k är lutningen och m är hur mycket linjen är förskjuten på y-axeln.

Om du har två punkter kan du räkna ut lutningen

$$k = \frac{y_2 - y_1}{x_2 - x_1}$$

Konstanten m får vi genom att sätta in värdet x = 0 i ekvationen, det vill säga y-värdet där linjen skär y-axeln.

Du har två punkter (50, 20) och (40, 90).

1. Beräkna räta linjens ekvation för dessa två punkter.
2. Rita ett koordinatsystem, sätt ut de två punkterna samt rita kurvan för ekvationen.

8. Område: Sannolikhet och statistik

8.1. Projekt: Sannolikhet

I matematik anger P(A) sannolikheten för att händelsen A inträffar:

P(A) = antalet gynnsamma utfall/antalet möjliga utfall

Exempelvis om du slår en 6-sidig tärning:

$$P\left(chansen\,att\,det\,blir\,en\,6:a\right)=\frac{1}{6}$$

För att Python ska kunna skapa slumptal måste du använda dig av modulen random.

För att använda en modul skriver du import följt av modulens namn, exempelvis

```
import random
```

som låter oss använda oss av slumptalsmodulen random.

Därefter kan du skapa olika slumptal genom flera olika funktioner. Om vi vill skapa ett slumpvärde i form av ett heltal mellan 1 och 6 skriver du

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

Du kan också stoppa in slumptalet i en variabel

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

1. Skriv ett program som slår en tärning 100 gånger och håller reda på hur många gånger det blir en 1:a, 2:a och så vidare. Skriv ut resultaten som totaler och som procent. Tips: Skapa 6 olika variabler som håller reda på hur ofta det blir en 1:a, en 2:a och så vidare.

2. Visa fördelningen från 1-6 i ett stolpdiagram.

8.2. Projekt: Träddiagram

Skriv ett program som konstruerar ett träddiagram utgående från att du slinglar slant. Lägg allt i en funktion som har antalet kast som argument.

Exempel (A = krona, B = klave)

(A: 1/2) (B: 1/2)

(A+A: 1/4) (A+B: 1/4) (B+A: 1/4) (B+B: 1/4)

(A+A+A: 1/8) (A+A+B: 1/8) ...

8.3. Projekt: Cirkeldiagram

(Innan detta projekt ska du har gått igenom Python: Lektioner: Listor.)

Du har 5-10 värden i en lista.

1. Utifrån denna lista ska du skapa en tabell med de olika värdenas relativa frekvens.

2. Skapa även ett cirkeldiagram med frekvenserna som grund.

Läs mer om hur du kan använda turtle.circle() för att göra tårtbitar: https://docs.python.org/3.3/library/turtle.html?highlight=turtle#turtle.circle.

8.4. Projekt: Undersöka slumptal

För att få ditt program att fungera olika varje gång du kör det kan du använda dig av slumptal.

För att Python ska kunna skapa slumptal måste du använda dig av ett tillägg till språket. I Python kallas sådana tillägg för moduler och det finns en mängd olika moduler för olika saker, exempelvis statistik, grafik och så vidare.

För att använda en modul skriver du import följt av modulens namn, exempelvis

```
import random
```

som låter oss använda oss av slumptalsmodulen random.

Om vi vill skapa ett slumpvärde i form av ett heltal mellan 1 och 6 skriver du

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

Du kan också spara slumptalet i en variabel

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

Ett program för att skriva ut 10 slumptal kan se ut så här:

```
import random
raknare = 0
while (raknare < 10):
    slumptal = random.randint(1,6)
    print(slumptal)
    raknare = raknare + 1
```

Nu kan vi undersöka slumpen! Hur många procent av alla slag blir en 6:a?

```
import random
raknare = 0
traff = 0
antal_slag = 100
while (raknare < antal_slag):
    slumptal = random.randint(1,6)
    if slumptal == 6:
        traff = traff + 1
    raknare = raknare + 1

print("Andel 6or: ")
print(traff / antal_slag)
```

1. Hur många slag borde bli en 6:a teoretiskt?

2. Hur många slag måste du upp i för att det alltid ska bli som i teorin?

3. Om du hinner, ändra programmet så att det också håller reda på antal 1:or, 2:or osv och skriver ut statistik även för de siffrorna!

8.5. Projekt: Undersöka slumptal med en ”dictionary”

En funktion som skapar slumptal ska vara just slumpmässig. Men hur kan en maskin som en dator som endast gör vad vi ber den om klara av detta?

Nu ska vi undersöka Pythons slumpfunktion!

För att kunna göra det är det en bra idé att använda en datastruktur i Python som kallas för ”dictionary” (uppslagsverk).

Ett uppslagsverk består av par som består av en nyckel och ett värde. I vårt fal är nyckeln antal ögon på tärningen, medan värdet är hur många gånger tärningen kommit upp med sådana ögon.

Exempel:

slumptal = {1: 46}

om tärningen har kommit upp med en 1:a 46 gånger.

Att skapa ett uppslagsverk:

```
slumptal = {}
```

Du bör därefter nollställa uppslagsverket för tärningsslag 1 till 6:

```
raknare = 0
while raknare < 6:
    raknare = raknare + 1
    slumptal[raknare] = 0
```

Om du vill öka räknaren för ett slumptal:

```
tal = random.randint(1,6)
slumptal[tal] = slumptal[tal]+1
```

1. Skriv ett program som skapar 1000 slumptal från 1 till 6 och lägger in deras frekvens i uppslagsverket.
2. Skriv ut antalet som en 1:a, 2:a och så vidare förekom.
3. Skriv ut förekomsten som ett procenttal.
4. Räkna ut medelvärdet.
5. Prova att öka antalet kast med tärningen från 1000 till något högre och se om frekvensen blir mer jämnt fördelad.

Gör gärna efterforskningar på nätet om hur en dator skapar slumpvärden. Vad gör random.seed()?

8.6. Projekt: Diagram

Genomför Projekt: Undersöka slumptal och presentera information i form av stapeldiagram som du gör med Turtle.

Lägg ritandet av en stapel i en funktion så att du kan återanvända programkoden.

Inspiration: http://interactivepython.org/runestone/static/thinkcspy/Functions/ATurtleBarChart.html

8.7. Projekt: Spridningsmått och lägesmått

1. Gör en undersökning i din klass för att skapa ett underlag till denna övning. Undersökningen kan gälla exempelvis föräldrarnas ålder.

2. Skriv ett program som räknar ut olika mått: Medelvärde, median, variationsbredd, kvartiler.

3. Skapa ett lådagram som visar resultaten av din undersökning.

Läs mer om begreppen: https://www.matteboken.se/lektioner/matte-2/statistik/kvartiler-och-ladagram

8.8. Projekt: Kombinatorik

Om du vill veta på hur många olika sätt du kan kombinera 3 förrättar, 5 huvudrätter och 2 efterrätter så använder du multiplikation:

$$3 \cdot 5 \cdot 2 = 30$$

Skriv ett program som visar alla möjliga kombinationer av ovanstående meny. Kom på egna namn på rätterna och skriv ut deras namn. Tips: Använd for eller while!

9. Avancerade projekt

9.1. Projekt: Pascals triangel

Läs på om Pascals triangel: https://sv.wikipedia.org/wiki/Pascals_triangel

1. Skriv ett program som skriver ut Pascals triangel i form av text.
2. Skriv ett program som skriver ut Pascals triangel med hjälp av Turtle.

9.2. Projekt: Romerska siffror

De romerska siffrorna byggde på bokstäver i olika kombinationer.

III = 3

IV = 4

Romerska	**I**	**V**	**X**	**L**	**C**	**D**	**M**
Tal	1	5	10	50	100	500	1000

Regler:

1) Lika bokstäver efter varandra ska adderas.
2) Om en mindre siffra står före en större ska den subtraheras.
3) Står en mindre siffra efter en större ska den adderas.
4) Ett mindre tal som sätts före ett större tal måste vara minst 1/10 av det större talet.

   MCMXLVI = 1946

Skriv en funktion som tar en sträng med romerska siffror och omvandlar den till ett tal. Regel 4 krånglar till lösningen, så vänta gärna med den.

Läs mer om romerska siffror: https://sv.wikipedia.org/wiki/Romerska_siffror

9.3. Projekt: Mastermind

Ta reda på hur spelet Mastermind fungerar.

1. Skriv ett program som låter dig spela Mastermind.
2. Skriv ett program som spelar Mastermind om du hittar på en lösning.

9.4. Projekt: Game of Life

Game of life är en klassisk simulering: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

Studera reglerna för Game of Life och skriv ett program som kan utföra simuleringen. Tips!

Implementera genom att ha två spelbräden med nuvarande läge (state) och nästa läge.

10. Python: Lektioner

Här följer ett antal exempel och lektioner för att förklara, förtydliga och motivera olika programmeringstekniker.

10.1. Två olika typer av slingor

Eftersom en stor styrka hos en dator är förmågan att snabbt upprepa saker är slingor en viktig byggsten.

Python har två grundläggande sätt att skapa slingor, while och for.

Båda dessa två program gör samma sak.

Med while:

```
tal = 1
while tal <= 100:
    print(tal)
    tal = tal + 1
```

Med for:

```
for tal in range(1, 101):
    print(tal)
```

I denna bok har jag favoriserat while-konstruktionen för att jag anser att den är lättare att ta till sig. For-slingan har fördelar som visar sig först när Python används på ett mer avancerat sätt, bortom denna bok. Dessutom finns for i många andra programmerings-språk men fungerar där på ett annat sätt, vilket skulle kunna vara förvirrande.

Om du som lärare finner att du gillar for mer är du välkommen att modifiera och komplettera instruktionerna.

10.2. Variabler och slinga

Låt användaren träna multiplikationstabeller.

```
tabell = int(input("Ange multiplikationstabell:"))

faktor = 0
antal_ratt = 0
upp_till = 10

while faktor < upp_till:
    faktor = faktor + 1
    ratt_svar =  faktor * tabell
    print("Vad blir", faktor, "*", tabell)
    gissning = int(input("Svar:"))
    if gissning == ratt_svar:
        print("Rätt!")
        antal_ratt = antal_ratt + 1
    else:
        print("Fel! Rätt svar är", ratt_svar)
print("Du hade", antal_ratt, "rätt av", upp_till)
```

10.3. Funktioner, slingor och test

Detta program skriver ut några egenskaper för ett tal som användaren matar in.

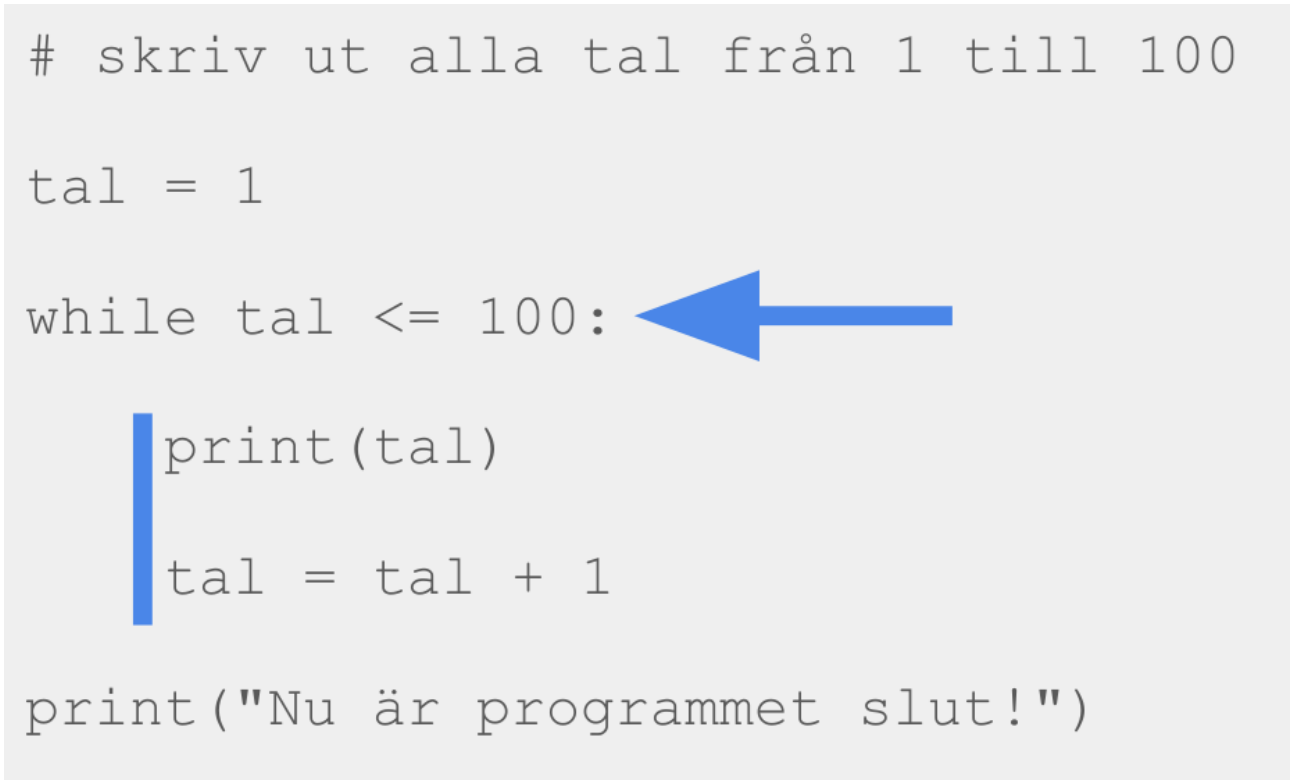
```
def jamnt(x):
    kvot = tal / 2
    if kvot == int(kvot):
        return True
    else:
        return False

def udda(x):
    return not jamnt(x)

def primtal(x):
    for i in range(2, x):
        if x % i == 0:
            return False
     return True

tal = int(input("Vilket tal vill du undersöka?"))

if jamnt(tal):
    print("Talet är jämnt.")

if udda(tal):
    print("Talet är udda.")

if primtal(tal):
    print("Talet är ett primtal.")
else:
    print("Talet är inte ett primtal.")
```

10.4. Indrag och block

Exempel 1:

```
tal = int(input("Skriv in ett tal mellan 1 och 10:"))
if tal < 1 or tal > 10:
    print("Det är inte mellan 1 och 10!")
else:
    print("Bra gjort!")
    print("Ditt tal är:", tal)
print("Här fortsätter programmet oavsett värdet du skrev in!")
```

Exempel 2:



10.5. Turtle

Det finns olika sätt för Python att skapa grafik. Det som är enklast att komma igång med heter turtle (sköldpadda). Det har sitt ursprung i 60-talet och tidigare språk för att lära unga att programmera.

Turtle är en liten sköldpadda som rör sig över skärmen. Överallt där den rör sig lämnar den ett spår. Genom att styra sköldpaddan kan du rita olika geometriska former och figurer.

För att kunna arbeta med turtle måste du först importera det:

```
import turtle
```

Så här ser ett enkelt program

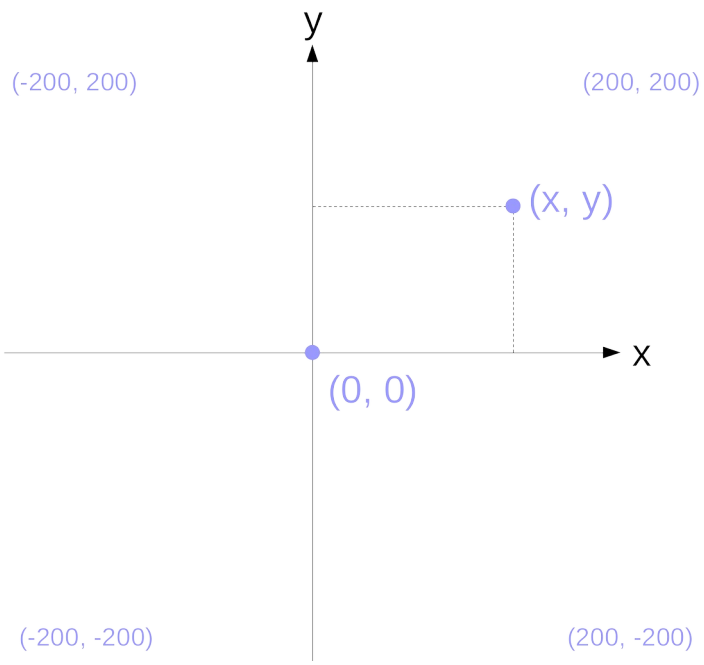
```
import turtle
padda = turtle.Turtle()
padda.forward(50)
padda.left(90)
padda.forward(50)
```

Normalt ser den lilla sköldpaddan ut som en pil. Men du kan byta utseende genom:

```
padda.shape("turtle")
```

I stället för "turtle" kan du ange “arrow”, “circle”, “square”, “triangle”, “classic”.

Turtle använder sig av följande koordinatsystem:



Skriv ett program som använder sig av Turtle. Låt det:

1. Rita en kvadrat.

2. Rita en rektangel.

3. Rita en femhörning.

4. Rita en åttahörning

Om du vill rita två eller flera likadana figurer så kan du använda while.

```
import turtle
padda = turtle.Turtle()
raknare = 0
while raknare < 4:
    raknare = raknare + 1
    # rita kvadrat
    padda.forward(50)
    padda.left(90)
    padda.forward(50)
    padda.left(90)
    padda.forward(50)
    padda.left(90)
    padda.forward(50)
    padda.left(90)
    # vrid nästa kvadrat
    padda.left(30)
```

Ännu snyggare blir det om vi ”bryter ut” kvadratritandet och lägger i en egen funktion:

```
import turtle
padda = turtle.Turtle()

def ritaFigur():
    padda.forward(50)
    padda.left(90)
    padda.forward(50)
    padda.left(90)
    padda.forward(50)
    padda.left(90)
    padda.forward(50)
    padda.left(90)

raknare = 0
while raknare < 3:
    raknare = raknare + 1
```

```
    # rita kvadrat
    ritaFigur()
    # vrid nästa kvadrat
    padda.left(30)
```

Observera att funktionen ritaFigur() måste ligga innan du använder den i programmet.

1. Skriv en funktion som ritar ett angivet antal kvadrater som roterar hela varvet runt. Lägg ritandet av kvadraten i en egen funktion.

Exempel: Om antalet kvadrater ska vara 12 så ska vinkeln mellan varje kvadrat vara 360 / 12 = 30 grader.

2. Skriv en funktion som ritar ett angivet antal trianglar som går hela varvet runt.

Kan du skriva uppgift 1 och 2 så att du bara behöver ändra i funktionen ritaFigur()?

3. Skriv en funktion som ritar cirklar som blir större och större. Du ska kunna ange hur många cirklar du vill att programmet ska rita. Avståndet mellan cirklarna får du själv komma fram till.

Du skapar en cirkel med hjälp av turtle.circle(radie).

Turtle har flera funktioner för att göra snyggare figurer. Här är några. Pröva dem!

- turtle.color(<färg>) - bestämmer sköldpaddans färg. Du kan hitta alla färgnamn du kan använda här: https://trinket.io/docs/colors eller https://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm
- turtle.pensize(<storlek>) - bestämmer linjens tjocklek. Ange en siffra.
- turtle.reset() - suddar din figur.
- turtle.write(<text>) - skriver text.

Se Projekt: Baertling för mer information om hur du arbetar med färger.

Läs mer om turtle: https://docs.python.org/3/library/turtle.html

10.6. Turtle, slingor och funktioner

Att rita en kvadrat:

```
import turtle
padda = turtle.Turtle()

def ritaFigur(sida):
    padda.forward(sida)
    padda.left(90)
    padda.forward(sida)
    padda.left(90)
    padda.forward(sida)
    padda.left(90)
    padda.forward(sida)
    padda.left(90)

ritaFigur(100)
```

Ett ännu snyggare sätt att rita en kvadrat är:

```
def ritaFigur(sida):
    for i in range(4):
        padda.forward(sida)
        padda.left(90)
```

Koden blir kortare. Det blir dessutom lättare att förändra koden till att rita exempelvis en sexhörning:

```
def ritaFigur(sida):
    for i in range(6):
        padda.forward(sida)
        padda.left(60)
```

Nu kan vi enkelt generalisera koden till att rita en n-hörning:

```
def ritaFigur(sida, n):
    vinkel = 360 // n # heltalsdivision
    for i in range(n):
        padda.forward(sida)
        padda.left(vinkel)
```

10.7. Slumptal

För att Python ska kunna skapa slumptal måste du använda dig av modulen random.

För att använda en modul skriver du import följt av modulens namn, exempelvis

```
import random
```

som låter oss använda oss av slumptalsmodulen random.

Därefter kan du skapa olika slumptal genom flera olika funktioner. Om vi vill skapa ett slumpvärde i form av ett heltal mellan 1 och 6 skriver du

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

Du kan också stoppa in slumptalet i en variabel

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

Eftersom datorer inte kan utgå från något slumpmässigt, de är förutbestämda att göra samma sak varje gång (vilket är deras styrka), så bör vi egentligen ge dem ett unikt frö att utgå ifrån varje gång.

Det gör vi med hjälp av funktionen

```
random.seed()
```

Python använder då datorns klocka som slump-frö, vilket brukar vara ok i våra sammanhang.

Gå gärna igenom detta inför 4.3, 4.9, 5.7 och 8.1.

10.8. Slumptal - skapa egna

Hur kan en dator, som ju alltid utför något som är förutsägbart, skapa slumptal?

En variant är att använda decimalerna i pi.

En annan är formeln

$$x_{n+1} = (a \cdot x_n + b) \bmod c$$

det vill säga nästa slumptal räknas ut från ett föregående värde genom multiplikation och addition. Modulus-operationen räknar ut resten och ser till att slumptalet hålls inom intervallet (0, c).

Variablerna a, b och c är givna heltal som ska vara ”väl valda”.

På svenska brukar denna formel kallas för linjär kongruensgenerator.

```
# Skapa 10 slumptal från 0 till 999
a = 781
b = 387
c = 1000
# startvärde, "frö"
x = 7
for raknare in range(1, 10):
    x = (a*x + b) % c
    print(x)
```

1. Använd denna funktion i Projekt: Undersöka slumptal och studera hur den beter sig.

2. Kan du hitta andra värden på a och b som skapar bra slumptal?

3. Kan du hitta värden på a och b som inte fungerar?

Läs mer:

- http://www.math.kth.se/matstat/gru/godis/slumptalgen.pdf
- http://www.maths.lth.se/matstat/kurser/fms120/markov.pdf
- https://sv.wikipedia.org/wiki/Linj%C3%A4r_kongruensgenerator

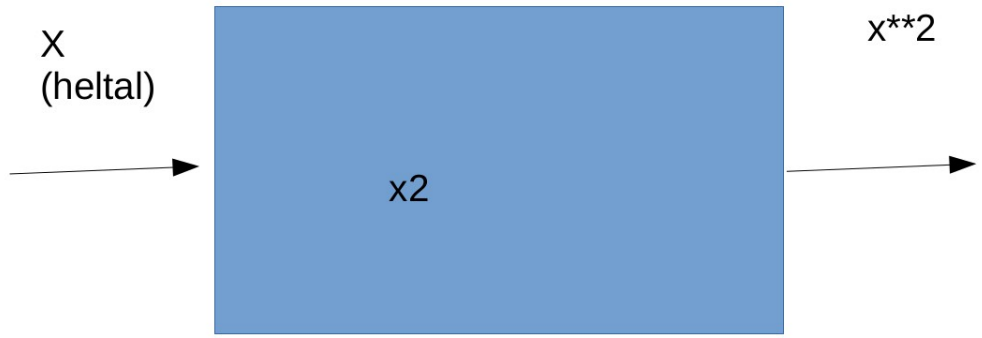
10.9. Flaggor

Så här skriver du program som utför olika saker varannan gång. Det använder sig av en så kallad flagga som skiftar mellan True och False.

```
# varannan gång
flagga = True

for i in range(1, 11):
    if flagga:
        flagga = False
        print("Udda!")
    else:
        flagga = True
        print("Jämnt!")
```

10.10. Funktioner

Denna lektion visar hur funktioner skapas och används, samt hur de kan användas för att räkna ut Pythagoras sats.




```
def x2(x):
  return x**2

print(x2(4))
```

```
# behövs för sqrt
import math

# använd en inbyggd funktion som räknar ut roten ur
print(sqrt(9))

# skapa funktionen x upphöjt till 2
def x2(x):
    return x**2

# använd funktionen direkt
print(x2(5))

# använd funktionen
tal = int(input("Skriv in ett tal du vill kvadrera:"))
print(x2(tal))

# funktion som använder en tidigare skapad funktion
```

```
def x4(x):
    return x2(x2(x))

def hypotenusan(a, b):
    c = math.sqrt(x2(a) + x2(b))
    return c

a = int(input("k1:"))
b = int(input("k2:"))
print(hypotenusan(a, b))
```

Här följer ett elevprojekt där användaren anger värden för katetrarna och hypotenusan. Det värde som söks anges som 0.

De här raderna kan ges som mall:

```
def pythagoras(a, b, c):
    if a == 0:
        # gör ngt
    elif b == 0:
        # gör ngt
    elif c == 0:
        # gör ngt
    else:
        # vad ska vi göra här???
```

Förslag på lösning.

```
import math

# skapa funktionen
def pythagoras(k1, k2, h):
    if k1 == 0:
        k1 = math.sqrt(h**2 - k2**2)
    elif k2 == 0:
        k2 = math.sqrt(h**2 - k1**2)
    elif h == 0:
        h = math.sqrt(k1**2 + k2**2)
    return k1, k2, h

# använd funktionen
a, b, c = pythagoras(0, 4, 5)
print(a, b, c)
```

10.11. Listor

Om vi ska spara exempelvis koordinater för att kunna rita en graf så behöver vi ett bra sätt att kunna lagra dem.

I Python finns något som kallas listor. De används på följande sätt:

```
>>> primtal = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
>>> primtal
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Vi har nu en lista över de tio första primtalen. De är numrerade från 0 till 9, vilket kallas för de olika värdenas index, så

```
>>> primtal[3]
7
```

Lägg till det 11:e primtalet:.

```
>>> primtal.append(31)
```

Leta efter ett värde och skicka tillbaka dess index:

```
>>> primtal.index(11)
4
```

Ta bort värden:

```
>>> primtal.remove(17)
```

eller för att ta bort det sista

```
>>> primtal.pop()
```

Du kan sortera en lista:

```
>>> primtal.sort()
```

Antal värden i listan:

```
>>> len(primtal)
```

Om du vill göra något med alla värden i en lista:

```
for tal in primtal:
    print(tal)
```

Läs mer om listor och andra datastrukturer:
https://docs.python.org/3/tutorial/datastructures.html

Vi har tre punkter:

(10, 11) - x-koordinaten 10 och y-koordinaten 11

(20, 21)

(30, 31)

Då kan vi skapa en lista bestående av tre listor med en x- och en y-koordinat:

```
>>> graf = [[10,11], [20,21], [30,31]]
>>> graf
[[10, 11], [20, 21], [30, 31]]
>>> len(graf)
3
>>> graf[1]
[20, 21]
>>> for punkt in graf:
...   x = punkt[0]
...   y = punkt[1]
...   print(x,y)
10 11
20 21
30 31
```

10.12. Variablers omfång

Variabler har ett omfång där deras värde gäller. Vi kan enklast studera det genom ett exempel.

```
def summa(a, b):
    c = 4
    print("Nu är vi i funktionen.")
    print("Variabeln a har värdet", a)
    print("Variabeln c har värdet", c)
    return a+b

a = 1
c = summa(2, 3)

print("Nu är vi utanför funktionen.")
print("Variabeln a har värdet", a)
print("Variabeln c har värdet", c)
```

I en funktion kan variabler alltså existera ”lokalt” och ha ett värde som bara finns inuti funktionen.

Det finns med andra ord två variabler med namnet a och c. Två globala som har ett värde utanför funktionen. Två lokala som har ett värde inuti funktionen summa().

Visa gärna vad som händer om ett argument ändras inne i funktionen.

(Detta avsnitt skulle kunna vara mycket mer omfattande men här ges bara några grunder. Utforska gärna på egen hand.)

10.13. Inbyggd matematik

10.13.1. Funktioner

Python har en inbyggt modul math som alltså inte behöver importeras för att kunna användas.

Läs mer om math: https://docs.python.org/3/library/math.html

10.13.2. Moduler

Python har många moduler. Några är extra intressanta ur ett matematikperspektiv:

- SciPy (https://www.scipy.org/) som är ett hem till bland andra SymPy och Matplotlib.
- SymPy (http://www.sympy.org/en/index.html) som låter dig hantera uttryck symboliskt och lösa ekvationer.

10.14. Felhantering och säker inmatning

När vi arbetar med input så kontrollerar vi aldrig om användaren matar in ett värde som är ok. Att mata in ett värde som inte är ok kan betyda att användaren skriver in bokstäver när vi förväntar oss ett tal, att de skriver in tal som är för stora eller för små.

Detta kan hanteras genom att varje input() följs av if-rader som kontrollerar svaret. Men det kan också handla om att använda Pythons inbyggda felhantering för att fånga upp fel: https://docs.python.org/3/tutorial/errors.html.

10.15. Läsa och skriva filer

I denna bok arbetar vi bara med data som matas in av användaren, skapas av programmet själv eller finns i listor.

Ofta hämtar dock program in data från filer som är lagrade på datorn: https://docs.python.org/3/tutorial/inputoutput.html.

10.16. Rekursion

Rekursion är ett kraftfullt verktyg som dock kan vara svårt att förstå. Men många matematiska områden blir riktigt eleganta med rekursion.

På dessa sidor finns redogörelser för hur Fibonaccis talföljd kan uttryckas på olika sätt inklusive rekursion:

- https://stackoverflow.com/questions/18009817/fibonacci-sequence-python
- https://stackoverflow.com/questions/13826810/fast-fibonacci-recursion

```
def iterative_fib(num):
    a = 0
    b = 1
    i = 0
    while i < num:
        a_prev = a
        a = b
        b = a_prev + b
        i = i + 1
    return a

def pythonic_fib(num):
    a = 0
    b = 1
    i = 0
    while i < num:
        a, b, i = b, a + b, i + 1
    return a

def recursive_fib(num):
    if num == 0:
        return 0
    elif num == 1:
        return 1;
    else:
        return recursive_fib(num - 1) + recursive_fib(num - 2)
```

11. Lärardel

Boken bygger på Python 3.

Dispositionen bygger på en vanligt förekommande disposition i läromedel i matematik. Tanken är att programmeringen inte ska bli en engångsinsats utan ett stående inslag och kontinuerligt utvecklingsområde för såväl pedagoger som elever.

Programmering ska inte vara ett självändamål. Programmering ska vara ett verktyg för att undersöka och arbeta med matematiska frågeställningar.

Så här skriver Skolverket i LGR11:

> *Genom undervisningen i ämnet matematik ska eleverna sammanfattningsvis ges förutsättningar att utveckla sin förmåga att*
>
> - *formulera och lösa problem med hjälp av matematik samt värdera valda strategier och metoder,*
> - *använda och analysera matematiska begrepp och samband mellan begrepp,*
> - *välja och använda lämpliga matematiska metoder för att göra beräkningar och lösa rutinuppgifter,*
> - *föra och följa matematiska resonemang, och*
> - *använda matematikens uttrycksformer för att samtala om, argumentera och*
> - *redogöra för frågeställningar, beräkningar och slutsatser.*

Att arbeta med programmering innebär i bästa fall att arbeta med alla dessa förmågor. Inte minst tror jag på programkod som ett verktyg för kommunikation.

Här är några av mina argument för att arbeta med programmering i matematik och teknik:

- Ett bra och spännande sätt att utveckla ämnet.
- Ett sätt att arbeta ämnesintegrerat och nyskapande genom att till exempel samarbeta teknik-slöjd.
- Ett sätt att utforska matematiska frågeställningar som inte låter sig göras på andra sätt.
- Ett sätt att fånga (vissa) elevers intresse genom att närma sig deras vardag. Det finns gott om programmeringsuppgifter med snabb och kontinuerlig återkoppling.
- Visa att undervisningen "hänger med" i utvecklingen.
- Därför att vi måste -- ändringar i skolans styrdokument.
- Att förvandla eleverna från konsumenter till producenter (av digitala produkter).
- Demokrati: Utan inblick i programmering kommer eleverna inte att förstå både

aktuella och framtida frågeställningar.

11.1. Så här kan du använda boken

Om du inte kan programmera i Python gå då först själv igenom kapitel 3.

Börja varje lektion med att gå igenom de nya begrepp som introduceras. Till din hjälp finns det förslag på färdiga lektioner som kan användas för genomgångar (kapitel 10).

Ge eleverna gott om tid att arbeta själva och uppmuntra dem att prova olika varianter, även sådant som inte explicit är omnämnt. Introduktionen (kapitel 3) och alla projekt (Kapitel 4-8) är skrivna som för att eleverna ska kunna arbeta självständigt.

> Varje avsnitt avslutas med små uppgifter som eleverna ska lösa. De är angivna inom rutor. Elevuppgifterna i projekten är även de angivna i rutor.

11.1.1. Planering - introduktion

Lektion 1 - Introduktion del 1 (2 timmar)

1. Hur du kör programkod (3.1)
2. Att köra Python rad för rad (3.2)
3. Matematiska uttryck (3.2)
4. Variabler (3.2)
5. Utskrift - print() (3.2)

Det är viktigt att påpeka att citat-tecken runt text är den raka varianten. På en del datorer (särskilt ipad) är det lätt hänt att det blir böjda citat-tecken. (Ipad: Håll kvar fingret på citat-tecknet så kan du välja den raka varianten.)

Lektion 2 - Introduktion del 2 (2 timmar)

6. Inmatning - input() (3.3)
7. Jämförelser - if (3.3)
8. Block (3.3)

Det är viktigt att poängtera hur indrag skapar block och hur de hänger ihop med kolon efter exempelvis if, for och while. Använd Lektion - Indrag och block.

Lektion 3 - Slingor (2 timmar)

1. Slingor - while (3.4)
2. Slingor - for (3.4)

Använd Lektion - Två olika typer av slingor samt Lektion - Variabler och slinga.

Lektion 4 - Funktioner (2 timmar)

1. Funktioner (3.5)
2. Funktioner med returvärden (3.5)

Använd Lektion - Funktioner, slingor och test och Lektion - Funktioner.

En funktion kan skicka tillbaka flera värden:

```
def summaochprodukt(x, y):
    return x + y, x * y

tal1 = 4
tal2 = 5
summa, produkt = summaochprodukt(tal1, tal2)
print("Summan av", tal1, "och", tal2, "är", summa)
print("Produkten av", tal1, "och", tal2, "är", produkt)
```

11.1.2. Planering - projekt

Nu kan eleverna börja med projekten. Eftersom tanken är att programmeringen ska vara ett verktyg för matematikundervisningen är de indelade i matematiska områden enligt den modell som många läromedel använder. Det går att arbeta med områdena i godtycklig ordning. Undantaget är det första området, Område: Tal, som bör användas först då det har längre genomgångar.

Varje område består av olika projekt som innebär att eleven ska producera ett program. Ibland innehåller de en introduktion av varierande längd som ger dem de nödvändiga verktygen.

Varje område har tillräckligt många projekt för att täcka årskurs 7-9. Jag tror en hinner med ca 4 projekt per termin, 1-2 projekt per område. Projekten ligger i en ungefärlig progressionsordning i boken.

Projekten tar normalt 1-2 timmar.

Läs igenom projekten och gör dem själv innan eleverna får arbeta med dem. Då ser du också om det är ytterligare något begrepp som behöver gås igenom.

TIll din hjälp kan du använda de lektioner som finns i kapitel 10:

- Områdena Geometri och Sannolikhet och statistik: Lektion Funktioner, Lektion Turtle, Lektion Turtle, slingor och funktioner
- Projekt 4.3 (Gissa faktorer), 8.1 (Sannolikhet): Lektion Slumptal
- Projekt 6.6 (π): Lektion Flaggor
- Projekt 7.4 (Linjediagram): Lektion Listor

Följande lektioner räknar jag som överkurs:

- Kapitel 10: Lektion Slumptal - skapa egna

- Kapitel 10: Lektion Variablers omfång
- Kapitel 10: Lektion Rekursion

Det finns också några lektioner som bygger på att du först utforskar området själv. Även dessa räknar jag som överkurs:

- Lektion Inbyggd matematik
- Lektion Felhantering och säker inmatning
- Lektion Läsa och skriva filer

I slutet finns ett fåtal förslag på större projekt som kan användas som utmaningar för elever som är snabba.

11.2. Bedömning

Elevens arbete med de olika projekten kan ligga till grund för bedömning i matematik.

Poängtera gärna för eleverna vikten med att kommentera sin kod.

Det går också att pröva elevernas kunskaper på de ordinarie matematik-proven. Här följer ett exmpel på en uppgift som eleverna kan få.

11.2.1. Område: Tal

Tema: Star Trek

USS Voyager är anfallet av 100 Klingon-rymdskepp. Bara var fjärde skepp är dock verkligt, resten är projiceringar. Data börjar med att skriva två små program för att plocka ut de riktiga. Han slås dock ut av en kortslutning i kaffebryggaren och hinner bara skriva färdigt två slingor från 1 till 100:

```
# program 1
tal = 0
while tal < 100:
    tal = tal + 1
    print(tal)

# program 2
for tal in range(1, 101):
    print(tal)
```

Deanna Troi är (som vanligt) tvungen att rädda besättningen. Hjälp henne att modifiera programmen så att de endast skriver ut vart fjärde tal (1, 5, 9 osv).

Bedömning av förmågor: 2 x E kommunikation, 2 x E problemlösning.

11.2.2. Exempel på prov

I appendix 17 finns ett exempel på ett prov jag gav samt facit.

11.2.3. Summativ bedömning

Så här sammanställde jag elevernas resultat på tre projekt samt provet.

Projekt Läxläsning	E	tot	C	tot	A	tot
Begrepp						
Kommunikation		1		1		
Metod						
Resonemang						
Problemlösning		1		1		

Projekt 4.1 Multiplikation och division	E	tot	C	tot	A	tot
Begrepp						
Kommunikation		2		2		1
Metod						
Resonemang						
Problemlösning		2		2		1

Projekt 4.2 Udda och jämna tal	E	tot	C	tot	A	tot
Begrepp						
Kommunikation		4		4		
Metod						
Resonemang						
Problemlösning		4		4		

Prov	E	tot	C	tot	A	tot
Begrepp		2				
Kommunikation		7		7		2
Metod		1		1		
Resonemang						
Problemlösning		7		7		2

12. Lösningar till uppgifter

Här följer en del lösningar och skisser till projekten.

12.1. Område: Tal

12.1.1. Projekt: Multiplikation och division

```
taljare = int(input("Skriv in täljaren:"))
namnare = int(input("Skriv in nämnaren:"))
kvot = 0
rest = taljare
while rest >= namnare:
    rest = rest - namnare
    kvot = kvot + 1
print ("Kvoten är:", kvot, "resten är:", rest)
```

12.1.2. Projekt: Jämna och udda tal

För att skapa udda tal kan vi ta vilket tal som helst,multiplicera det med 2 och lägga till 1. Vi kan också subtrahera 1 från ett jämnt tal.

```
print("Udda tal")
for tal in range(1, 51):
    print (tal * 2 - 1)
```

12.1.3. Projekt: Gissa faktorer

```
# Gissa faktorer

import random

ratt = 0
fel = 0

while ratt < 10:

    slumptal = random.randint(1, 100)

    print("Gissa två faktorer till ", slumptal, ".")

    faktor1 = int(input("Faktor 1:"))
    faktor2 = int(input("Faktor 2:"))

    if faktor1 * faktor2 == slumptal:
        ratt = ratt + 1
        print("Rätt!")
    else:
```

```
        fel = fel + 1
        print ("Fel!", faktor1, "*", faktor2, "=", faktor1 * faktor2,
".")


print ("Du gissade fel", fel, "gånger.")
```

12.1.4. Projekt: Problemlösning med råstyrka

12.1.5. Projekt: Faktorisering

Fundera på: Har du några ideer på hur programmet skulle kunna undvika att pröva alla tal? Du behöver inte skriva någon kod, bara beskriva något eller några sätt!

- Ja, multipler av tidigare tal verkar onödiga.
- Om det är ett jämnt tal så kan vi hoppa över alla jämna tal.

12.1.6. Projekt: Funktioner

12.1.7. Projekt: Funktioner som ger svar

12.1.8. Projekt: Primtal

12.1.9. Projekt: Slumptal

12.1.10. Projekt: Talbaser

```
bas = int(input("bas:"))
tal = int(input("tal:"))
i = 16 # vi arbetar med högst 16 bitar
while (i > 0):
    bit = int(tal / bas ** i)
    print(bit)
    tal = tal - (bit * (bas ** i))
    i = i - 1
```

Du kan skriva ut en varibels värde utan att det följs av en radbrytning genom

```
print(bit, end='')
```

12.2. Område: Algebra

12.2.1. Projekt: Euklides algoritm

Euklides algoritm effektivare:

```
def sgd2(x, y):
    while y > 0:
        if (x > y):
            x = x % y
        else:
            y = y % x
    return x

print(sgd2(15, 6))
```

Eftersom det första talet måste vara större (vi skickar ju tillbaka x, och kollar bara på om y > 0):

```
def sgd2(x, y):
    if (y > x):
        return sgd2(y, x)
    while (y > 0):
        if (x > y):
            x = x % y
        else:
            y = y % x
    return x
```

12.2.2. Projekt: Ekvationslösning med intervallhalvering 1

Svar: Hantera annat än heltal; flera nollpunkter; intervallets omfång.

12.2.3. Fibonacci

	0	**1**	**1**	**2**	**3**	**5**	**8**
0	a	b	a+b				
1		a	b	a+b			
2			a	b	a+b		
3				a	b	a+b	
4					a	b	a+b

12.2.4. Projekt: Kryptografi

```
klartext = "MATEMATIK"
kryptotext = ""
for bokstav in klartext:
    krypto_bokstav = chr(ord(bokstav)+1)
    kryptotext = kryptotext + krypto_bokstav
```

12.3. Område: Geometri

12.3.1. Projekt: Månghörning

```
import turtle
padda = turtle.Turtle()
padda.speed("slowest")

def ritaFigur(horn):
    # horn > 2
    vinkelSumma = (horn - 2) * 180
    vinkelSteg = int(vinkelSumma / horn)
    vinkel = 180-vinkelSteg
    print(vinkelSumma, vinkel)
    for i in range(0, vinkelSumma, vinkelSteg):
        padda.forward(20) # hur långt den ska rita beror på antal hörn
        padda.left(vinkel)

ritaFigur(24)
```

12.3.2. Projekt: Månghörning

12.3.3. Projekt: Rektangelns omkrets och area

```
import turtle
padda = turtle.Turtle()
padda.speed("fastest")

def ritaRektangel(bredd, hojd):
    padda.forward(bredd)
    padda.left(90)
    padda.forward(hojd)
    padda.left(90)
    padda.forward(bredd)
    padda.left(90)
    padda.forward(hojd)
    padda.left(90)

maxArea = 0

for b in range(10, 100, 10):
    h = 100-b
```

```
    ritaRektangel(b, h)
    area = b * h
    print("Rektangel: ", b, h, "Area: ", area)

    if area > maxArea:
        maxArea = area

print(maxArea)
```

Ett ännu snyggare sätt att rita en rektangel är

```
def ritaRektangel(bredd, hojd):
    for i in range(2):
        padda.forward(bredd)
        padda.left(90)
        padda.forward(hojd)
        padda.left(90)
```

Och om det är en kvadrat

```
def ritaKvadrat(sida):
    for i in range(4):
        padda.forward(sida)
        padda.left(90)
```

Lösningen kan även se ut så här:

```
import turtle
padda = turtle.Turtle()
#padda.speed("fastest")

def ritaFigur(bredd, hojd):
    padda.forward(bredd)
    padda.left(90)
    padda.forward(hojd)
    padda.left(90)
    padda.forward(bredd)
    padda.left(90)
    padda.forward(hojd)
    padda.left(90)

maxArea = 0
```

```
for b in range(10, 100, 10):
    h = 100 - b

    ritaFigur(b, h)

    area = b * h
    print("Kvadrat: ", b, h, "Area: ", area, "Omkrets:", 2 * (b + h))

    if area > maxArea:
        maxArea = area

print(maxArea)
```

12.3.4. Projekt: Digitalkamera

12.3.5. Projekt: Baertling

12.3.6. Projekt: π

```
# Gottfried Leibniz formel
# pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - ...

namnare = 3
summa = 1
flagga = False

while (namnare < 1000000):
    if flagga == False:
        summa = summa - 1/namnare
        flagga = True
    else:
        summa = summa + 1/namnare
        flagga = False

    namnare = namnare + 2

print(summa*4)
```

12.3.7. Projekt: Cirkelns omrets och area

```
import turtle
import math
```

```
padda = turtle.Turtle()

def ritaKvadrat(omkrets):
    sida = omkrets / 4
    for i in range(4):
        padda.forward(sida)
        padda.left(90)

def ritaCirkel(omkrets):
    diameter = omkrets / math.pi
    padda.circle(diameter/2)

x = 500
ritaKvadrat(x)

# centrera cirkeln
padda.up()
padda.forward(x / 4 / 2)
padda.right(90)
padda.forward(x / 4 / 2 / 4)
padda.left(90)
padda.down()

ritaCirkel(x)
```

12.3.8. Projekt: Volym

12.3.9. Projekt: Skala

12.3.10. Projekt: Pythagoras sats

12.3.11. Projekt: Rymddiagonal i kub

```
# Rymddiagonal

import math

def pythagoras(a, b):

    c = math.sqrt(a**2 + b**2)

    return c
```

```
x = int(input("Kubens sida: "))

d1 = pythagoras(x, x)
d2 = pythagoras(d1, x)

print ("Rymddiagonalen är ", d2)
```

12.4. Område: Samband och förändring

12.4.1. Projekt: Procent

12.4.2. Projekt: Lån

12.4.3. Projekt: Koordinatsystem

```
import turtle
padda = turtle.Turtle()

def ritaKoordinatsystem(bredd, hojd):
    padda.penup()
    padda.goto(-bredd, 0)
    padda.pendown()
    padda.goto(bredd, 0)
    padda.penup()
    padda.goto(0, -hojd)
    padda.pendown()
    padda.goto(0, hojd)

def ritaPunkt(x, y):
    padda.penup()
    padda.goto(x, y)
    padda.pendown()
    padda.dot(5)



ritaKoordinatsystem(200, 200)
ritaPunkt(50, -50)
ritaPunkt(50, 50)
```

12.4.4. Projekt: Linjediagram

12.4.5. Projekt: Stolp- och stapeldiagram

12.4.6. Projekt: Proportionalitet och linjära samband

12.4.7. Projekt: Kast av boll

```
import turtle
padda = turtle.Turtle()
```

```
g = 10
skala = 1.2

def ritaKoordinatsystem(bredd, hojd):
    padda.penup()
    padda.goto(-bredd, 0)
    padda.pendown()
    padda.goto(bredd, 0)
    padda.penup()
    padda.goto(0, -hojd)
    padda.pendown()
    padda.goto(0, hojd)

def ritaPunkt(x, y):
    padda.penup()
    padda.goto(x, y / skala)
    padda.pendown()
    padda.dot(5)

def kastparabel(t):
    return 1 + 5 * t - (g * (t ** 2)) / 2

ritaKoordinatsystem(200, 200)

for t in range(0, 200):
    y = kastparabel(t / 100)
    ritaPunkt(t, y * 100)
```

12.4.8. Projekt: Räta linjens ekvation

y = 2x + 10

12.5. Område: Sannolikhet och statistik

12.5.1. Projekt: Undersöka slumptal

```
import random

slumptal = {}
slag = 1000

raknare = 0
while raknare < 6:
    raknare = raknare + 1
    slumptal[raknare] = 0

raknare = 0
while raknare < slag:
    raknare = raknare + 1
    tal = random.randint(1, 6)
    slumptal[tal] = slumptal[tal] + 1

raknare = 0
while raknare < 6:
    raknare = raknare + 1
    print(slumptal[raknare], slumptal[raknare]/slag)
```

13. Resurser

13.1. Litteratur

- Doing Math With Python av Amit Saha: https://doingmathwithpython.github.io/pages/about.html
- Övningar för gymnasiet: https://wikiskola.se/index.php?title=Kategori:Python

13.2. Handledningar

- http://interactivepython.org/courselib/static/thinkcspy/index.html
- http://www.pythontutor.com
- https://www.py4e.com/

14. Appendix: Programmeringsmiljö

Hur väljer du en miljö att koda i?

Aspekter

- Det ska fungera på olika plattformar, ipad mm.
- Elever ska kunna spara kod.
- Elever ska kunna lämna in kod för bedömning.

Kandidater där vi kan programmera i Python i en webbläsare:

- https://repl.it/
- http://jupyter.org/
- http://www.skulpt.org/
- https://www.pythonanywhere.com/
- https://trinket.io/
- http://www.codeskulptor.org
- https://snakify.org/

För skolor som har en fri miljö finns fler alternativ: https://www.python.org/downloads/.

15. Appendix: Kodstil

Python ska helst skrivas enligt en given mall: https://www.python.org/dev/peps/pep-0008/.

I korthet:

- indrag ska vara 4 blanksteg
- kommentarer ska inledas med # och vara på en egen rad
- matematiska operatorer (+, - med flera) ska omges av blanksteg, men inte inuti parenteser
- komma-tecken ska följas av blanksteg
    - *a = f(1, 2) + g(3, 4)*

16. Appendix: Python och C++

Jag använder Python för matematik och C/C++ för teknik (Arduino).

För att underlätta för mina elever på högstadiet har jag därför skrivit denna guide. Därför är språket också inriktat på resultat och inte exakthet.

16.1. Ett exempelprogram

Python har fler inbyggda funktioner från start.

I C/C++ behöver du ofta inkludera ett bibliotek som ger dig tillgång till nya funktioner.

Python hanterar block med indrag medan C/C++ använder krullparenteser.

16.1.1. Python

```
# ett program skrivet i Python 3
i = int(input("Ange ett heltal:"))
print("Värdet du angav var", i, "och dess dubbla värde är", i * 2)

if (i == 5):
    print("Talet är fem!")
```

16.1.2. C++

Du måste först skapa variabler, inkludera bibliotek. Sen ligger huvudprogrammet i en funktion kallad ”main”.

```
// ett program skrivet i C++
#include <iostream>
using namespace std;

int main ()
{
    int i;
    cout << "Ange ett heltal: ";
    cin >> i;
    cout << "Värdet du angav var " << i;
    cout << " och dess dubbla värde är " << i*2 << ".\n";

    if (i == 5) {
        cout << "Talet är fem!";
    }

    return 0;
}
```

16.2. Variabler

I Python måste du inte skapa variabler innan du använder dem. Du kan också spara olika typer av värden i dem, text, tal etc.

C++ är strängare: Du måste skapa variabeln innan du använder den och du måste tala om vilken typ av värde du ska spara i den.

16.2.1. Python

```
tal = 45
summa = tal + 96

tal = "fyrtiofem"
```

16.2.2. C/C++

```
int tal;
int summa;
// kan också skrivas som int tal, summa;

tal = 45;
summa = tal + 96;

tal = "fyrtiofem"; // ger felmeddelande
```

Du kan också skriva såhär om du vet vad variabeln ska ha för start-värde:

```
int tal = 45;
```

16.3. In- och utmatning

16.3.1. Python

```
tal = int(input("Ange ett tal:"))
```

16.3.2. C/C++

```
#include <iostream>
using namespace std;

int main ()
{
    int tal;
    cout << "Ange ett tal: ";
    cin >> tal;

    return 0;
}
```

16.4. Slingor (loopar)

16.4.1. Python

While:

```
# skriv ut alla tal från 1 till 100
tal = 1
while tal <= 100:
    print(tal)
    tal = tal + 1
```

For:

```
# skriv ut alla tal från 1 till 100
for tal in range(1, 101):
    print(tal)
```

Notera att Pythons for skiljer sig från for i C++. Pythons for upprepar egentligen kod över en samling (itererar) vilket är kraftfullt men utanför denna bok, som enbart använder sig av for på ett liknande sätt som i C++ med hjälp av range().

16.4.2. C/C++

While:

```
# skriv ut alla tal från 1 till 100
int tal = 1;

while (tal <= 100) {
    cout << tal;
    tal++; // öka tal med 1
}
```

For:

```
# skriv ut alla tal från 1 till 100
int tal;

for (tal = 1; tal <= 100; tal++) {
      cout << tal << endl;
   }
```

Do:

```
do {
    cout << tal << endl;
    tal++;
} while (tal <= 100);
```

16.5. Övrigt

16.5.1. Kommentarer

Python använder # för att skriva en kommantar på en rad.

C++ har två olika metoder:

- // används för kommentar på en rad
- /* och */ kan användas för att kommentera över flera rader

16.5.2. Slumptal

Python:

```
# skapa ett slumptal från 1 till 100
import random
slumptal = random.randint(1,100)
```

C++:

```
// skapa ett slumptal från 1 till 100
#include <stdlib.h>
int slumptal;
slumptal = rand() % 100 + 1;
```

17. Appendix: Exempel på prov

17.1. Prov

Programmering i Matematik - Prov 1

Ditt namn: _____

Begrepp

Begrepp: 2E

Vad är ett program?

Vad är en variabel?

Matematik

Problemlösning: E Kommunikation: E

Skriv i Python hur du räknar ut fyra gånger fem, därefter delat med två. (Resultatet ska bli tio.)

Räknaren

Problemlösning: 2E Kommunikation: 2E

Följande program ska skriva ut alla tal från 1 till 10. Rätta felen i programmet!

```
räknare = 0
while räknare < 10:
    print(räknare)
```

Räkna ut kvoten

Problemlösning: E C Kommunikation: E C

Skriv ett program som

- låter användaren mata in två heltal
- skriver ut kvoten av de två talen som ett decimaltal.

Udda tal

Problemlösning: E C Kommunikation: E C Metod: E C

Skriiv ett program som

- låter användaren mata in ett tal (max)
- räknar upp alla udda tal från 1 till max.

Frågesport

Problemlösning: 2E 2C 2A Kommunikation: 2E 2C 2A

Skriv ett program som

- skapar två slumptal mellan 1 och 10
- låter användaren ge svaret på vad deras produkt blir
- håller reda på när användaren svarar rätt
- med hjälp av en slinga upprepar steg 1-3 fem gånger
- innan det avslutas skriver ut hur många rätt användaren hade

Du skapar slumptal genom:

```
import random
slumptal = random.randint(1, 10)
```

17.2. Facit

Matematik

Problemlösning: E Kommunikation: E

Skriv i Python hur du räknar ut fyra gånger fem, därefter delat med två. (Resultatet ska bli tio.)

```
>>> (4*5)/2
```

Räknaren

Problemlösning: 2E Kommunikation: 2E

Följande program ska skriva ut alla tal från 1 till 10. Rätta felen i programmet!

```
räknare = 0
while räknare < 10:
    print(räknare)
```

Rättat:

```
räknare = 1
while räknare < 11:
    print(räknare)
    räknare = räknare + 1
```

Räkna ut kvoten

Problemlösning: E C Kommunikation: E C

Skriv ett program som

- låter användaren mata in två heltal
- skriver ut kvoten av de två talen som ett decimaltal.

```
taljare = int(input("Täljare:"))
namnare = int(input("Nämnare:"))
print("Kvoten är", taljare / namnare)
```

Alternativ:

```
taljare = int(input("Täljare:"))
namnare = int(input("Nämnare:"))
kvot = 0
```

```
while taljare >= namnare:
  taljare = taljare - namnare
  kvot = kvot + 1
print("Kvoten är:", kvot)
print("Resten är:", taljare)
```

Udda tal

Problemlösning: E C Kommunikation: E C Metod: E C

Skriiv ett program som

- låter användaren mata in ett tal (max)
- räknar upp alla udda tal från 1 till max.

```
max = int(input("Räkna upp till:"))
raknare = 1
while raknare <= max:
    print(raknare)
    raknare = raknare + 2
```

eller

```
max = int(input(“Skriv in övre gränsen:”))
for räknare in range(1, max + 1, 2):
    print(räknare)
```

Frågesport

Problemlösning: 2E 2C 2A Kommunikation: 2E 2C 2A

Skriv ett program som

- skapar två slumptal mellan 1 och 10
- låter användaren ge svaret på vad deras produkt blir
- håller reda på när användaren svarar rätt
- med hjälp av en slinga upprepar steg 1-3 fem gånger
- innan det avslutas skriver ut hur många rätt användaren hade

```
import random
```

```
resultat = 0

raknare = 0

while (raknare < 5):
  slumptal1 = random.randint(1, 10)
  slumptal2 = random.randint(1, 10)
  produkt = slumptal1 * slumptal2
  print("Vad blir", slumptal1, " * ", slumptal2)

  gissning = int(input("Svar:"))

  if gissning == produkt:
    resultat = resultat + 1
    print("Rätt!")
  else:
    print("Fel! Rätt svar är ", produkt)

  raknare = raknare + 1

print("Du hade", resultat, "rätt.")
```

18. Kopieringsunderlag

18.1. Python sammanfattning

Python sammanfattning

Staffan Melin 2021

Vad	Exempel
ge variabel ett värde	`antal = 5` `namn = "Staffan"`

skriv ut variabel	`print(antal * 10)`
	`print(namn)`
	`print("Antal är", antal)`

öka variabel med ett	`antal = antal + 1`
göra beräkningar	`omkrets = 3.14 * diameter`
mata in text	`namn = input("Skriv in ditt namn:")`
mata in heltal	`antal = int(input("Skriv in ett tal:"))`

testa ett värde	`if antal == 11:`
	`  print("Fler än 10!")`
	`else:`
	`  print("Färra än 10!")`

testa flera värden	`if namn == "Staffan" or namn == "Malin":`
	`  print("Du är mattelärare.")`
	`elseif namn == "Kajsa":`
	`  print("Du är spanska-lärare.")`
	`else:`
	`  print("Du är lärare i något annat.")`

upprepa något (while)	`tal = 1`
	`while tal <= 100:`
	`  print(tal)`
	`  tal = tal + 1`

upprepa något (for)	`for tal in range(1, 101):` `  print(tal)`

skapa en funktion	`def jamforTal(a, b):`
	`  if a > b:`
	`  print("Det första talet är störst!")`
	`  elif a < b:`
	`  print("Det andra talet är störst!")`

använda en funktion	`jamforTal(16, 15)`