Grundläggande programmering

1 Inledning

Övningarna i Programmeringsteknik under vecka 1–3 av läsperiod ht1 är avsedda för dig som inte har programmerat tidigare. Under övningarna får du en kort introduktion till programmering och lär dig en del terminologi.

Avsikten är att du ska arbeta med materialet på egen hand eller tillsammans med en kamrat. Övningsledaren kommer att hjälpa till om du får problem och kommer att förklara svåra saker vid tavlan. Om du inte hinner med alla uppgifterna under en övning bör du försöka lösa uppgifterna på egen hand och kontrollera dina lösningar mot lösningförslagen som finns sist i häftet.

Rekommenderad takt:

Vecka 1	Avsnitt 1–6.2
Vecka 2	Avsnitt 6.3–6.5 (till och med uppgift 14)
Vecka 3	Avsnitt 6.5 (uppgift 15)–6.6

2 Datorer och datorprogram

En dator

- har en *processor* (CPU) som kan utföra instruktioner,
- har ett *primärminne* där man lagrar program som exekveras (”körs”) och de data som programmet arbetar med,
- har ett *sekundärminne* där man kan spara program och data i filer,
- kan *kommunicera* med användaren via tangentbord, mus och skärm.

Ett program som körs i datorn består av mycket enkla instruktioner av typen ”hämta ett tal från en viss minnesplats”, ”lagra ett tal i en minnesplats”, ”addera två tal”, ... Det finns också instruktioner för att jämföra talvärden med varandra och hoppa till en annan instruktion i programmet om jämförelsen får ett visst resultat. Uppsättningen av instruktioner som datorn ”förstår” kallas datorns *maskinspråk*. Man programmerar normalt inte i maskinspråk, eftersom det är tidsödande och risken att göra fel är stor. I stället utnyttjar man *högnivåspråk* som till exempel Java, C, C# eller Visual Basic.

Ett program i högnivåspråk består också av instruktioner till datorn, men instruktionerna är lättare för människor att skriva och att läsa. Innan man kan köra ett högnivåspråksprogram måste det översättas (kompileras) till instruktioner i datorns maskinspråk. Översättningen görs av ett program, en *kompilator*.

Exempel på instruktioner i ett Javaprogram:

```
interest = balance * interestRate / 100 * days / 360;
System.out.print("Räntan är " + interest + " kr.");
```

Instruktionerna beräknar räntan (`interest`) på ett visst belopp (`balance`) med en given räntesats (`interestRate`) under ett givet antal dagar (`days`). Därefter skrivs räntan ut på skärmen.

Även om man aldrig tidigare har sett ett Javaprogram kan man kanske förstå en del av detta. Formeln för ränteberäkning känner man nog igen. På den andra raden känner man igen ordet print (”skriv ut”) och ser att man skriver ut räntan och en del text.

Uppgift 1 Läs igenom avsnitt 1.1 i läroboken, om datorer och datorprogram. Det räcker att du läser översiktligt så att du blir bekant med innehållet.

Uppgift 2 Förklara följande termer: CPU, primärminne, sekundärminne. Operativsystem, program, maskinspråk, högnivåspråk, kompilator.

3 Programmering

Programmering är inte bara att skriva instruktioner i ett programspråk. Programmering är ”problemlösning” där det gäller att formulera en lösning till ett givet problem. Om problemet är enkelt (att beräkna räntan på ett bankkonto) så är det lätt att hitta lösningen, om problemet är svårt (att bestämma vilket drag som ska göras i ett schackprogram) så är det svårt att hitta lösningen. Vi kommer att syssla med enkla problem. Lösningarna till de enkla problemen förekommer som byggstenar i lösningarna till svåra problem.

En *algoritm* är en beskrivning av lösningen till ett problem. Man kan uttrycka algoritmer i vilket språk som helst.

Exempel Algoritm för att addera två heltal:

- Addera siffra för siffra bakifrån. För över minnessiffra om det behövs.

Man kan uttrycka algoritmen noggrannare:

- Gå igenom talen bakifrån. Så länge det finns siffror kvar i något av talen:
  - Addera två siffror och den eventuella minnessiffran från föregående addition.
  - Om summan är större än 9: minska summan med 10 och kom ihåg att vi har en minnessiffra.
  - Summan ger nästa siffra bakifrån i resultatet.

Den senare formuleringen är lätt att översätta till ett programspråk.

Uppgift 3 Formulera en algoritm för någonting vardagligt: att laga en cykelpunktering, steka pannkakor, ... Förfina de ingående instruktionerna tills de ligger på ”lagom” låg nivå.

Du kan här behöva utnyttja konstruktioner av typen ”Om något är uppfyllt så gör en viss sak, annars gör något annat” och eventuellt ”Så länge något inte är uppfyllt, gör en viss sak”. Dessa konstruktioner kallas alternativ- respektive repetitionskonstruktioner.

4 Data

Alla algoritmer arbetar med data av något slag. En algoritm för att addera tal arbetar med tre talvärden (talen som ska adderas och resultatet), en algoritm för att beräkna räntan på ett bankkonto arbetar med kontoställning, räntesats och antal dagar som ränteberäkningen gäller. I program lagrar man data i *variabler*. Varje variabel motsvaras av en minnesplats i datorn.

En variabel är en storhet som man själv inför och ger ett namn och som man kan tilldela värden. När man tilldelar en variabel ett värde raderas variabelns tidigare värde. Man kan ta reda på en variabels aktuella värde och utnyttja det i beräkningar när man så önskar.

En variabel är av en bestämd *typ* beroende på slaget av värden som kan tilldelas variabeln. I Java har man bland annat tillgång till följande typer:

Typ	I Java	Förklaring
Heltal	`int`	Variabler av heltalstyp kan tilldelas heltalsvärden som 27, 0, -2345, 10000.
Reellt tal	`double`	Variabler av reell typ kan tilldelas reella värden som 0.2, -1.578, $5.23 \cdot 10^{21}$. När man programmerar använder man decimalpunkt i stället för decimalkomma.

Det finns fler datatyper i Java, till exempel för tecken (`char`) och logiska värden (`boolean`).

Man måste *deklarera* alla variabler som man använder. Det innebär att man anger variablernas namn och typ. När man deklarerar en variabel passar man också på att ge variabeln ett startvärde.

Exempel I följande programrader deklarerar man två variabler. Texten efter `//` är kommentarer i programmet. I en kommentar kan man skriva godtycklig text.

```
int sum = 0;                // heltalsvariabel, startvärde 0
double interestRate = 4.25; // reell variabel, startvärde 4.25
```

Det är viktigt att man väljer namn som ger en uppfattning om vad variablerna betyder. Till exempel är `interestRate` ett bra namn i ett program för ränteberäkning. Programmet hade fungerat lika bra om variabeln hade haft namnet `xyz`, men det hade blivit mycket svårare att läsa.

I datorns minne motsvaras de båda variablerna av två minnesplatser:

`sum`	0
`interestRate`	4.25

5 Fullständiga program

I detta häfte ger vi många exempel där vi skriver ett antal lösryckta satser. För att det ska bli ett riktigt Javaprogram som man kan kompilera och köra måste man infoga satserna i en `main`-metod i en *klass*. Namnet på klassen väljer man själv.

I princip ska det se ut så här (`import`-satsen behövs bara om man i programmet läser data från tangentbordet; se avsnitt 6.2):

```
import java.util.Scanner;

public class Example {
    public static void main(String[] args) {
        // ... här infogar man satserna
    }
}
```

6 Satser

6.1 Inledning

I högnivåspråk kallar man instruktionerna man skriver för *satser*. Vi ska här studera de mest grundläggande satserna, till att börja med tilldelningssatser. Sedan beskriver vi satser för att uttrycka alternativ (`if`-satser) och för repetition (`for`-satser och `while`-satser).

6.2 Tilldelningssatser, inläsning och utskrift

I en tilldelningssats ger man en variabel ett nytt värde. Man skriver på följande sätt:

```
variabel = nytt värde;
```

Likhetstecknet utläser man ”tilldelas värdet”. När tilldelningssatsen utförs beräknas först det nya värdet, sedan tilldelas variabeln i vänsterledet det nya värdet. Variabelns tidigare värde raderas ut.

Det är viktigt att man förstår att tilldelningssatsen är en instruktion till datorn: satsen `a = b` betyder att man ger datorn ordern ”beräkna värdet av `b`, lägg detta värde i variabeln `a`”. Man kan senare ändra `a`:s värde, men om man därefter återigen vill ge `a` samma värde som `b` så måste man utföra satsen `a = b` en gång till. Ett tilldelningstecken är alltså inte något slags likhetstecken.

Det nya värdet i högerledet kan vara en konstant, som till exempel i satsen `sum = 0`, men det kan också vara ett mer komplicerat *uttryck* som talar om hur man ska beräkna det nya värdet, till exempel i satsen `sum = a + b + c`. Uttryck skriver man ungefär som i matematiken, till exempel skriver man sådant som ska beräknas först inom parentes. Man skriver multiplikation med tecknet `*`, division med `/`, addition med `+` och subtraktion med `-`.

Exempel på tilldelningssatser. Notera att alla satser avslutas med semikolon. Satserna utförs i tur och ordning.

```
int x = 10;         // x får värdet 10
int y = 3 * x + 1; // y får värdet 3 * 10 + 1 = 31
int z = y - x;      // z får värdet 31 - 10 = 21
x = x + 1;          // x får värdet 10 + 1 = 11, y och z
                    // ändras inte
```

När man ska följa exekveringen i ett program är det bra att göra upp en tabell över de ingående variablerna, där man anger variablernas aktuella värden. I exemplet får en sådan tabell följande utseende efter det att de fyra satserna utförts:

x	11
y	31
z	21

Före den sista tilldelningssatsen stod det 10 i `x`-rutan.

Uppgift 4 Betrakta följande programavsnitt:

```
int nbrA = 2;
int nbrB = nbrA + 3;
int nbrC = 2 * (nbrA + nbrB) + 1;
nbrA = nbrC / 3;
nbrC = 0;
```

Gör upp en tabell över variablerna och notera variablernas värden när tilldelningssatserna exekveras.

Uppgift 5 Skriv följande som tilldelningssatser i Java (alla variabler förutsätts vara deklarerade som `double`-tal):

y tilldelas $\frac{1}{2ab}$

y tilldelas $(a+b)^2(b+c)^2$

y tilldelas $1 + x + \frac{x^2}{2}$

De flesta moderna program har ett grafiskt användargränssnitt där man ger kommandon till programmet genom att trycka på knappar och välja ur menyer och där man matar in data i textrutor. Men ibland vill man läsa in värden från tangentbordet och skriva ut resultat som text på skärmen, och det gör man enligt följande exempel:

```
System.out.println("Skriv två tal");
Scanner scan = new Scanner(System.in);
int a = scan.nextInt();
int b = scan.nextInt();
int sum = a + b;
System.out.println("Summan av talen är " + sum);
```

Man läser heltal med `nextInt()`, reella tal med `nextDouble()`. Vad `Scanner` och `System.in` är förklarar vi senare i kursen.

`println` betyder ”print line” och innebär att det som står innanför parentesen skrivs ut. Nästa utskrift hamnar på ny rad. Observera citationstecknen när man skriver ut text: `println("sum")` betyder att de tre bokstäverna `s`, `u` och `m` i ordet `sum` skrivs ut; `println(sum)` betyder att värdet av variabeln `sum` skrivs ut.

Notera att inläsning och utskrift ses från datorns synpunkt: `int a = scan.nextInt()` betyder att datorn ska läsa ett värde (som vi skriver på tangentbordet), `System.out.println(sum)` betyder att datorn ska skriva ett värde (som vi kan läsa på skärmen).

Uppgift 6 Indata består av tre tal: en tid i timmar, minuter och sekunder. Skriv satser som läser talen och skriver ut tiden i sekunder. Det kan se ut så här:

```
Skriv tiden (timmar, minuter, sekunder):
2 10 43
Antalet sekunder: 7843
```

De vanliga matematiska funktionerna (kvadratrot, sinus, logaritm, ...) finns inbyggda i Java. När man ska beräkna ett funktionsvärde skriver man funktionens namn samt det värde, argumentet, för vilket funktionen ska beräknas. I programmering kallar man oftast ett argument för en *parameter* till funktionen.

I Java finns bland annat följande standardfunktioner:

`Math.sqrt(x)`	$\sqrt{x}$
`Math.sin(x)`	$\sin x$
`Math.cos(x)`	$\cos x$
`Math.exp(x)`	e^x
`Math.log(x)`	$\ln x$

När man ska beräkna sinus eller cosinus ska argumentvärdet vara uttryckt i radianer.

Exempel I följande satser läser man ett tal x från tangentbordet och beräknar motsvarande sinusvärde.

```
Scanner scan = new Scanner(System.in);
double x = scan.nextDouble();
double sinValue = Math.sin(x);
System.out.println("sin(" + x + ") = " + sinValue);
```

Om man inte behöver sinusvärdet senare i programmet kan man klara sig utan variabeln `sinValue`:

```
System.out.println("sin(" + x + ") = " + Math.sin(x));
```

Uppgift 7 Skriv följande i Java (alla variablerna är reella):

`y` tilldelas	$\sin 2 + 4\cos 3x$
`c` tilldelas	$\sqrt{a^2 + b^2}$
`x` tilldelas	$\dfrac{\ln(a-b)}{2\sin 2\pi c}$

6.3 Satser för alternativ och repetition

Om man bara har tillgång till tilldelningssatser kan man bara skriva mycket enkla program. För att man ska kunna skriva ”riktiga” program måste man ha satser för *alternativ*, dvs att man i programmet väljer mellan två olika saker att utföra och *repetition*, dvs att man utför samma sak många gånger.

I Java uttrycker man alternativ med `if`-satser, repetition med `for`-satser eller `while`-satser.

6.4 if-satser

En `if`-sats har följande utseende:

```
if (Villkor) {
    satser // utförs om villkoret är uppfyllt
} else {
    satser // utförs om villkoret inte är uppfyllt
}
```

Villkoret kan till exempel vara `x == 10` (uppfyllt om värdet av `x` är lika med 10, notera två likhetstecken) eller `a + b < 100` (uppfyllt om summan av `a` och `b` är mindre än 100). Man kan också använda `!=` (skilt från), <= (mindre än eller lika med), > och >=. Ibland inträffar det att man inte ska utföra någonting när villkoret inte är uppfyllt. Då utelämnar man `else`-grenen i satsen.

Exempel

```
Scanner scan = new Scanner(System.in);
double a = scan.nextDouble();
double b = scan.nextDouble();
if (a < b) {
    System.out.println("a är mindre än b");
} else {
    System.out.println("a är större än eller lika med b");
}
```

Uppgift 8 Skriv en Javasats som beräknar det största av talen `a` och `b` och lagrar värdet i variabeln `max`.

Ofta behöver man uttrycka mera komplicerade villkor som innehåller mer än ett led. Man kan då knyta ihop flera delvillkor med `&&`, som betyder ”och”, eller `||`, som betyder ”eller”. Villkoret `v1 && v2` är uppfyllt om båda delvillkoren `v1` och `v2` är uppfyllda, `v1 || v2` är uppfyllt om minst ett av delvillkoren `v1` och `v2` är uppfyllt.

Exempel Ett bussbolag säljer biljetter för 15 kr. Om man är yngre än 20 år eller äldre än 65 år får man 25% rabatt. Så här kan man beräkna biljettpriset:

```
double discount = 0.25;
double ticketPrice = 15.0;
if (age < 20 || age > 65) {
    ticketPrice = ticketPrice * (1 - discount);
}
```

Alternativ formulering, som skulle göra det enkelt att införa olika rabatter för barn och äldre:

```
double childDiscount = 0.25;
double elderlyDiscount = 0.25;
double ticketPrice = 15.0;
if (age < 20) {
    ticketPrice = ticketPrice * (1 - childDiscount);
} else if (age > 65) {
    ticketPrice = ticketPrice * (1 - elderlyDiscount);
}
```

När man ska testa flera villkor efter varandra skriver man det nya `if` direkt efter föregående `else`.

Uppgift 9 Sitt BMI (Body Mass Index) räknar man ut genom att dividera sin vikt (i kilo) med längden (i meter) i kvadrat. Enligt WHO ska BMI ligga mellan 18.5 och 25.0, annars är man under- eller överviktig.

Skriv satser som läser in vikt och längd och beräknar BMI. Dessutom ska ett omdöme skrivas ut, enligt följande exempel:

```
Skriv vikt (kg) och längd (cm):
80 170
BMI är 27.7
Övervikt enligt WHO
```

I stället för Övervikt kan det stå Undervikt eller Normalvikt.

Uppgift 10 Skriv Javasatser för följande uppgifter:

- Om `a` tillhör intervallet $[0, 10]$ ska `a` minskas med 1, annars ska `a` inte ändras.
- Talen `a` och `b` är båda $\neq 0$. Om `a` och `b` har samma tecken (dvs om båda är > 0 eller båda är < 0) ska `x` ökas med 10, annars ska `x` minskas med 10.

6.5 for-satser

Med en `for`-sats uttrycker man att något ska repeteras ett bestämt antal gånger. Det kan antingen vara så att antalet gånger är en konstant, till exempel 100, eller så att man i programmet kan räkna ut antalet gånger. När man ska utföra någonting 100 gånger skriver man så här:

```
for (int i = 0; i < 100; i++) {
    satser
}
```

Först sätts `i` $= 0$. Eftersom `i` är < 100 utförs satserna och `i` ökas med 1 (`i++` betyder detsamma som `i = i + 1`). `i` är fortfarande < 100, så satserna utförs igen, osv. Resultatet blir att satserna utförs för `i`-värdena 0, 1, 2, ..., 99. Man behöver inte börja räkna på 0, men det är praktiskt i många fall.

Exempel Läs in ett n-värde, skriv ut talen 0, 2, 4, ..., $2n$:

```
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
for (int i = 0; i <= n; i++) {
    System.out.println(2 * i);
}
```

Uppgift 11 Skriv satser som läser in ett heltal `n` och som producerar en tabell över de `n` minsta heltalen samt talens kvadrater och kvadratrötter. Exempel med `n` $= 4$:

```
1   1   1.000
2   4   1.414
3   9   1.732
4  16   2.000
```

Exempel I nedanstående satser läser man in ett `n`-värde. Sedan läser man in `n` stycken tal och kontrollerar att de alla är ≥ 0.

```
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
int nbrPositive = 0;
for (int i = 0; i < n; i++) {
    int nbr = scan.nextInt();
    if (nbr >= 0) {
        nbrPositive = nbrPositive + 1;
    }
}
if (nbrPositive == n) {
    System.out.println("Alla talen var positiva");
} else {
    System.out.println("Fel -- " + (n - nbrPositive) +
            " tal var negativa");
}
```

En vanligt förekommande uppgift är beräkning av summor av olika slag. Man ska alltså beräkna summan av ett antal termer, och man går igenom termerna i tur och ordning och adderar dem till summan. Man behöver åtminstone två variabler: en för att hålla reda på summan, en för den aktuella termen. Före summeringen måste man nollställa summavariabeln så att man har ett värde att börja med.

I nedanstående satser beräknas summan $1^3 + 2^3 + 3^3 + \ldots + 100^3$. I varje steg i repetitionen adderas en ny term (variabeln `term`) till summan (variabeln `sum`).

```
int sum = 0;
for (int i = 1; i <= 100; i++) {
    int term = i * i * i;
    sum = sum + term;
}
```

Uppgift 12 Skriv satser som läser 30 temperaturvärden och beräknar och skriver ut medeltemperaturen.

Uppgift 13 Skriv satser som läser ett n-värde och beräknar och skriver ut $n!$ (n-fakultet, $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$).

Uppgift 14 Indata består av 101 reella tal. Det första talet kallas `limit`. Skriv satser som läser talen och beräknar två summor: dels summan av de tal som är mindre än `limit`, dels summan av de tal som är större än `limit`. Tal som är lika med `limit` ska inte summeras.

Uppgift 15 Beräkna summan $1^2 - 2^2 + 3^2 - 4^2 + \ldots + n^2$ ($-n^2$ om n är jämnt). n-värdet ska läsas in i början av programmet. Ledning: utnyttja en variabel `sign` som omväxlande ges värdet $+1$ och -1. Multiplicera termen med `sign`.

Uppgift 16 Skriv satser som läser 10 tal och som kontrollerar att talen är sorterade i växande ordning. Om det finns fel i sorteringsordningen ska utskrift ske enligt nedanstående exempel. Talföljden:

```
1   1   2   7   4   5   12  11  12  23
```

ska ge utskriften:

```
Tal nr 5 är fel, dess värde är 4
Tal nr 6 är fel, dess värde är 5
Tal nr 8 är fel, dess värde är 11
```

Observera att 5-an är felsorterad, trots att den är större än 4. Man jämför med 7-an, som är det hittills största talet.

6.6 while-satser

Ett annat slag av repetition är ”så länge”-repetition där ett villkor avgör hur länge repetitionen ska hålla på. Man använder en `while`-sats:

```
while (Villkor) {
    satser
}
```

Om villkoret är uppfyllt utförs satserna, precis som i en `if`-sats. Men sedan kontrolleras villkoret igen — om det fortfarande är uppfyllt utförs satserna en gång till, varefter villkoret kontrolleras, osv. För att inte repetitionen ska fortsätta i all oändlighet måste det bland satserna som repeteras finnas en sats som åstadkommer att villkoret inte längre gäller.

Exempel I följande satser beräknas summan av ett antal positiva heltal som läses från tangentbordet. Talföljden avslutas med ett negativt tal.

```
int sum = 0;
Scanner scan = new Scanner(System.in);
int nbr = scan.nextInt();
while (nbr >= 0) {
    sum = sum + nbr;
    nbr = scan.nextInt();
}
```

Här hade det varit omöjligt att utnyttja en `for`-sats — man vet ju inte på förhand hur många positiva tal som kommer att läsas.

Uppgift 17 Ett arbete ger lön enligt följande: första dagen är lönen en krona. De följande dagarna får man dubbelt så mycket betalt som man fick närmast föregående dag. Skriv satser som avgör hur många dagar man måste arbeta innan man har tjänat ihop en miljon kronor.

Uppgift 18 Indata består av en följd av positiva heltal. Efter det sista talet finns ett negativt tal. Skriv satser som räknar hur många gånger som två intilliggande tal är lika.

Exempel: om talen är `3 4 4 7 2 1 1 1 9` ska resultatet bli 3 (`4 4`, `1 1`, `1 1` en gång till).

Uppgift 19 Om x är ett reellt tal kan man beräkna e^x genom att summera följande serie:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Skriv satser som läser ett x-värde och beräknar e^x genom att summera serien (i praktiken hade man skrivit `Math.exp(x)`). Man kan naturligtvis inte summera ett oändligt antal termer. I stället tar man i summan bara med termer som är större än ett tal ϵ (epsilon), som läses in först i programmet.

Ledning: beräkna varje ny term med hjälp av den föregående termen (därigenom undviker man ”overflow” vid fakultetsberäkningen). Till exempel beräknas $x^4/4!$ som (föregående term) $* \, x/4$, det vill säga $x^4/4! = x^3/3! * x/4$.

Lösningar till övningsuppgifterna

Observera: detta är *förslag* till lösningar. Man kan alltid lösa en uppgift på mer än ett sätt.

1. —

2. Definitionerna finns i läroboken och i avsnitt 2.

3. Som exempel tar vi instruktioner för att grädda pannkakor. Till 12 pannkakor behöver man 3 ägg, 6 dl mjölk, $2\,{}^{1}/_{2}$ dl mjöl, ${}^{1}/_{2}$ tsk salt. Beskrivning på hög nivå:

   1. Blanda alla ingredienserna.
   2. Grädda pannkakorna.

   En nybörjare behöver säkert noggrannare instruktioner (från ICA-kuriren):

   1. Vispa ägg, hälften av mjölken, hela mjölmängden och saltet till en slät, klimpfri smet. Tillsätt resten av mjölken och matfettet.
   2. Hetta upp en stekpanna. Smörj eventuellt laggen med lite smör till första pannkakan. Häll cirka 1 dl av smeten i stekpannan. Sänk värmen.
   3. Grädda tills ytan stelnat och blivit matt och undersidan fått fin guldgul färg.
   4. Vänd och grädda pannkakan på andra sidan.
   5. Rör om då och då i smeten under gräddningen, så att mjölet inte sjunker till botten.

   I matrecept brukar instruktionerna ligga på ungefär denna nivå och den är säkert ”lagom” för de flesta. Men man kan ha många invändningar:

   - Punkt 1: det står inte att man ska knäcka äggen och kasta skalen innan man vispar. Bland ingredienserna står det inte något om matfett. Man bör vispa även efter man tillsatt resten av mjölken och matfettet.
   - Punkt 2: hur varm ska stekpannan vara? Hur mycket är ”lite” smör? Hur mycket ska man sänka värmen?
   - Punkt 4: hur länge ska man grädda pannkakan på andra sidan?
   - Punkt 5: hur ofta är då och då?

   När en dator ska exekvera en algoritm måste man vara noggrann med alla detaljer.

4. Tabellen visar hur variabelvärdena ändras efter varje tilldelning:

`nbrA`	2	2	2	5	5
`nbrB`		5	5	5	5
`nbrC`			15	15	0

5. ```
   y = 1 / (2 * a * b);
   y = (a + b) * (a + b) * (b + c) * (b + c);
   y = 1 + x + x * x / 2;
   ```

6. ```
   System.out.println("Skriv tiden (timmar, minuter, sekunder):");
   Scanner scan = new Scanner(System.in);
   int hours = scan.nextInt();
   int minutes = scan.nextInt();
   int seconds = scan.nextInt();
   int totalSeconds = 3600 * hours + 60 * minutes + seconds;
   System.out.println("Antalet sekunder: " + totalSeconds);
   ```

7.
```
y = Math.sin(2) + 4 * Math.cos(3 * x);
c = Math.sqrt(a * a + b * b);
x = Math.log(a - b) / (2 * Math.sin(2 * 3.141592654 * c));
```

8.
```
double max;
if (a > b) {
    max = a;
} else {
    max = b;
}
```

9.
```
System.out.println("Skriv vikt (kg) och längd (cm):");
Scanner scan = new Scanner(System.in);
double weight = scan.nextDouble();
double height = scan.nextDouble() / 100;
double bmi = weight / (height * height);
System.out.println("BMI är " + bmi);
if (bmi < 18.5) {
    System.out.println("Undervikt enligt WHO");
} else if (bmi <= 25.0) {
    System.out.println("Normalvikt enligt WHO");
} else {
    System.out.println("Övervikt enligt WHO");
}
```

10.
```
if (a >= 0 && a <= 10) {
    a = a - 1;
}

if (a > 0 && b > 0 || a < 0 && b < 0) {
    x = x + 10;
} else {
    x = x - 10;
}
```

11.
```
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
for (int i = 1; i <= n; i++) {
    System.out.println(i + " " + (i * i) + " " + Math.sqrt(i));
}
```

12.
```
double sum = 0;
Scanner scan = new Scanner(System.in);
for (int i = 0; i < 30; i++) {
    double temperature = scan.nextDouble();
    sum = sum + temperature;
}
double averageTemperature = sum / 30;
System.out.println("Medeltemperatur: " + averageTemperature + " grader");
```

13.
```
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
int factorial = 1;
for (int i = 2; i <= n; i++) {
    factorial = factorial * i;
}
System.out.println(n + "! = " + factorial);
```

14.
```
Scanner scan = new Scanner(System.in);
double limit = scan.nextDouble();
double smallSum = 0;
double largeSum = 0;
for (int i = 0; i < 100; i++) {
    double nbr = scan.nextDouble();
    if (nbr < limit) {
        smallSum = smallSum + nbr;
    } else if (nbr > limit) {
        largeSum = largeSum + nbr;
    }
}
System.out.println("Summan av tal < " + limit + " = " + smallSum);
System.out.println("Summan av tal > " + limit + " = " + largeSum);
```

15.
```
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
int sum = 0;
int sign = +1;
for (int i = 1; i <= n; i++) {
    sum = sum + sign * i * i;
    sign = - sign;
}
System.out.println("Summan är " + sum);
```

16.
```
Scanner scan = new Scanner(System.in);
int currentMax = scan.nextInt();
for (int i = 1; i < 10; i++) {
    int nbr = scan.nextInt();
    if (nbr < currentMax) {
        System.out.println("Tal nr " + (i + 1) + " är fel, " +
                "dess värde är " + nbr);
    } else {
        currentMax = nbr;
    }
}
```

17.
```
int daySalary = 1;
int totalSalary = 0;
int days = 0;
while (totalSalary < 1000000) {
    totalSalary = totalSalary + daySalary;
    days = days + 1;
    daySalary = 2 * daySalary;
}
System.out.println("Man måste arbeta " + days + " dagar.");
```

18.
```
Scanner scan = new Scanner(System.in);
int previousNbr = scan.nextInt();
int adjacentEqual = 0;
if (previousNbr >= 0) {
    int currentNbr = scan.nextInt();
    while (currentNbr >= 0) {
        if (currentNbr == previousNbr) {
            adjacentEqual = adjacentEqual + 1;
        }
        previousNbr = currentNbr;
        currentNbr = scan.nextInt();
    }
}
System.out.println("Antal gånger två intilliggande tal var lika: " +
        adjacentEqual);
```

19.
```
Scanner scan = new Scanner(System.in);
double epsilon = scan.nextDouble();
double x = scan.nextDouble();
double sum = 0;
double term = 1;
int i = 0;
while (Math.abs(term) > epsilon) {
    sum = sum + term;
    i = i + 1;
    term = term * x / i;
}
System.out.println("exp(" + x + ") är ungefär " + sum);
```