IN1010

Repetisjon Uke 2

Java for deg som kan Python

(Basert på Dag Langmyhr og Stein Gjessings presentasjoner)

31. januar, 2025

Input

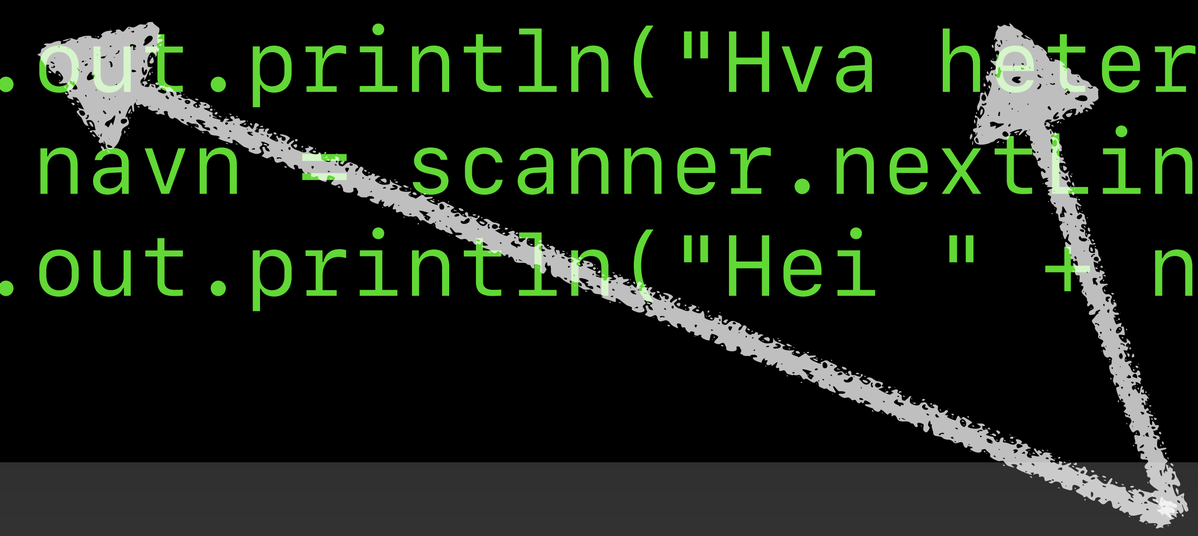
```
Scanner scanner = new Scanner(System.in);
System.out.println("Hva heter du? ");
String navn = scanner.nextLine();
System.out.println("Hei " + navn);
```



System.in gir oss inndata som en strøm fra terminalen

Input

```
Scanner scanner = new Scanner(System.in);
System.out.println("Hva heter du? ");
String navn = scanner.nextLine();
System.out.println("Hei " + navn);
```



Ved hjelp av klassen *Scanner*, kan vi hente teksten inn

Input

```
Scanner scanner = new Scanner(System.in);
System.out.println("Hva heter du? ");
String navn = scanner.nextLine();
System.out.println("Hei " + navn);
```

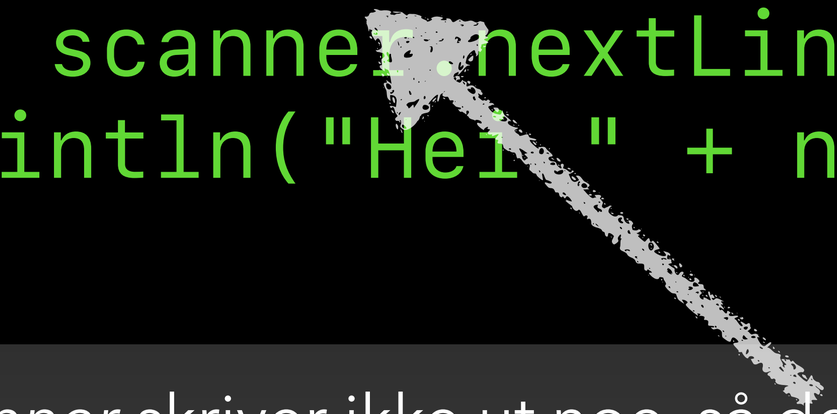


En av metodene vi kan bruke er *nextLine()*

Den henter en *String* ut fra neste linje som den får inn fra *System.in*

Input

```
Scanner scanner = new Scanner(System.in);
System.out.println("Hva heter du? ");
String navn = scanner.nextLine();
System.out.println("Hei " + navn);
```



Metodene til *Scanner* skriver ikke ut noe, så det må vi gjøre for seg

Fungerer ikke som Python sin *input*.

Importere klasser

```
import java.util.Scanner;

class HovedProgram {
   …
```

Scanner er en klasse som må importeres, men den tilhører standardbiblioteket, så man trenger ikke finne eksterne filer

Importere klasser

I Java så har vi en stor mengde med klasser som vi kan bruke

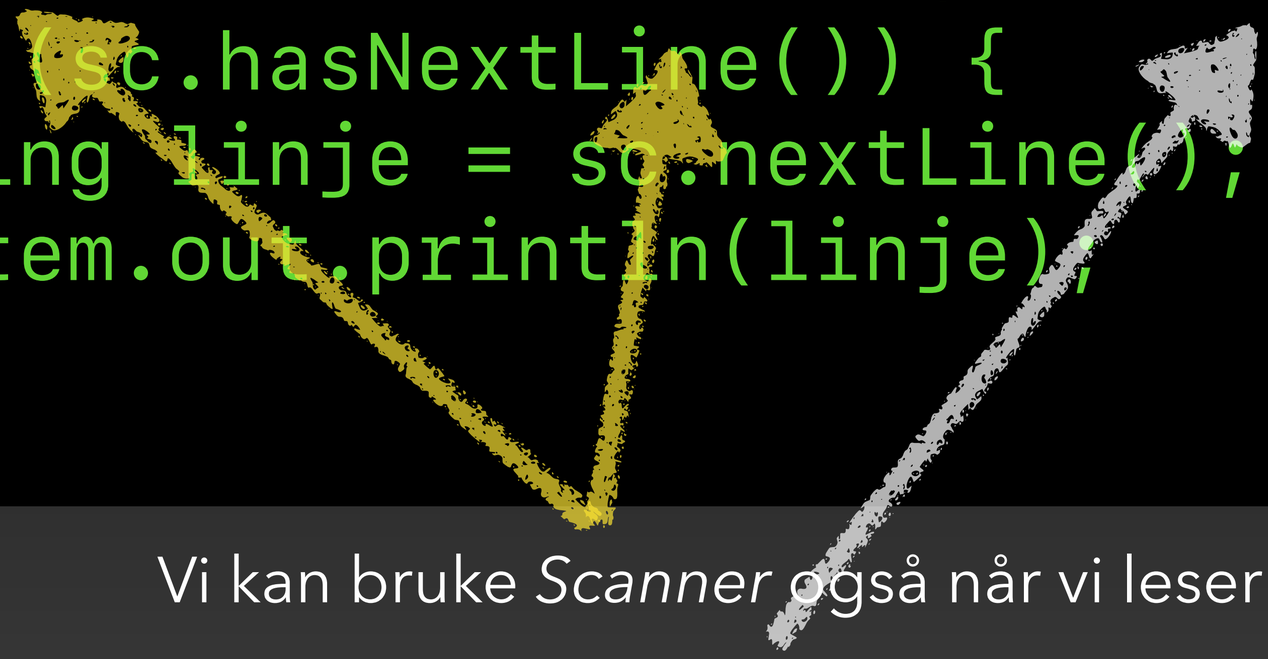
Og standardbiblioteket som følger med er dokumentert online

https://docs.oracle.com/javase/8/docs/api/

Det er ikke noe som man vil pugge, men slå opp i ved behov

Lese inn fra filer

```
Scanner sc = new Scanner(new File("hei.txt"));
while (sc.hasNextLine()) {
  String linje = sc.nextLine();
  System.out.println(linje);
}
```



Vi kan bruke *Scanner* også når vi leser inn fra filer.

I tillegg så bruker vi klassen *File* for å fortelle Java hvordan filen finnes og brukes

Lese inn fra filer

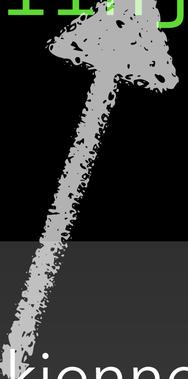
```
Scanner sc = new Scanner(new File("hei.txt"));
while (sc.hasNextLine()) {
   String linje = sc.nextLine();
   System.out.println(linje);
}
```



I tillegg så bruker vi her en annen metode fra *Scanner*, *hasNextLine()*, som forteller oss om det er flere linjer å lese

Lese inn fra filer

```
Scanner sc = new Scanner(new File("hei.txt"));
while (sc.hasNextLine()) {
  String linje = sc.nextLine();
  System.out.println(linje);
}
```



Og nextLine() kjenner vi igjen fra før

Unntak (Exceptions)

I en del tilfeller så vet vi *ikke* om det vi ønsker
å gjøre vil kunne gjennomføres.

F.eks. når vi skal lese inn en fil,
så kan det være at filen ikke eksisterer

Unntak (Exceptions)

```
try {
   Scanner sc = new Scanner(new File("fil.txt")
} catch (Exception e) {
   System.err.println("Feil med filen");
   System.exit(1);
}
```



Når vi har slike tilfeller, så prøver vi å "fange" unntaket ved hjelp av å "prøve" operasjonen vi vil gjøre.

Unntak (Exceptions)

Når vi kompilerer programmet, så vil den
si fra hvis vi prøver å gjøre noe som potensielt
ikke kan kjøres uten at et unntak
oppstår

I dokumentasjonen til standardbiblioteket, så
står det også hvilke unntak som kan oppstå

Unntak (Exceptions)

```
try {
   Scanner sc = new Scanner(new File("fil.txt")
} catch (FileNotFoundException | NullPointerException e) {
   System.err.println("Feil med filen");
   System.exit(1);
}
```



Det er lurt å eksplisitt oppgi de typene av unntak som vi ønsker å behandle

Ellers kan det være at andre typer unntak blir behandlet på en måte vi ikke ønsker

Programparametere

```
java MittProgram "min_mappe/fin_tekst.txt"
```



Det kan være praktisk å kunne oppgi informasjon til programmet

når vi starter det opp

Programparametere

```
public static void main(String[] args) {
    System.out.println(args[0]);
    File fil = new File(args[0]);
}
```



Det henter vi ut i main-metoden fra array-variabelen *args*.

Programparametere

```
public static void main(String[] args) {
    System.out.println(args[0]);
    File fil = new File(args[0]);
}
```



Da kan vi f.eks. skrive det ut i terminalen

Array element 0 er første parameteret

Programparametere

```
public static void main(String[] args) {
    System.out.println(args[0]);
    File fil = new File(args[0]);
}
```

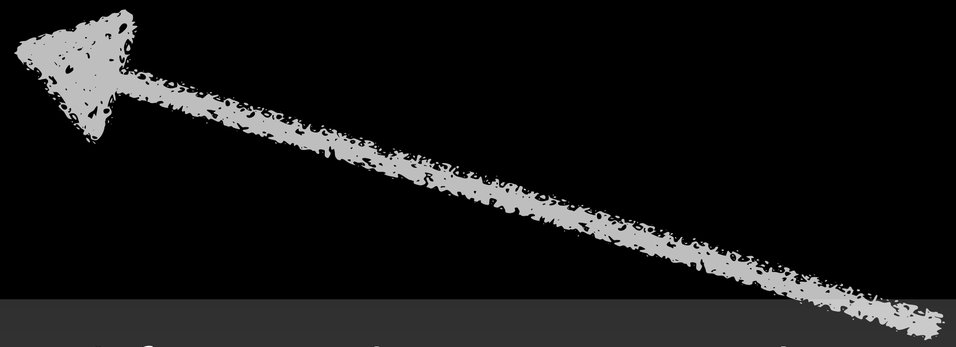


Eller vi kan bruke det for å lage et fil-objekt

Og på alle andre måter hvor vi kan bruke et String-objekt

Litt mer om arrays

```
args.length
```



For å finne ut hvor mange elementer et array har

I dette tilfellet kan vi finne ut hvor mange parametere programmet har

Litt mer om arrays

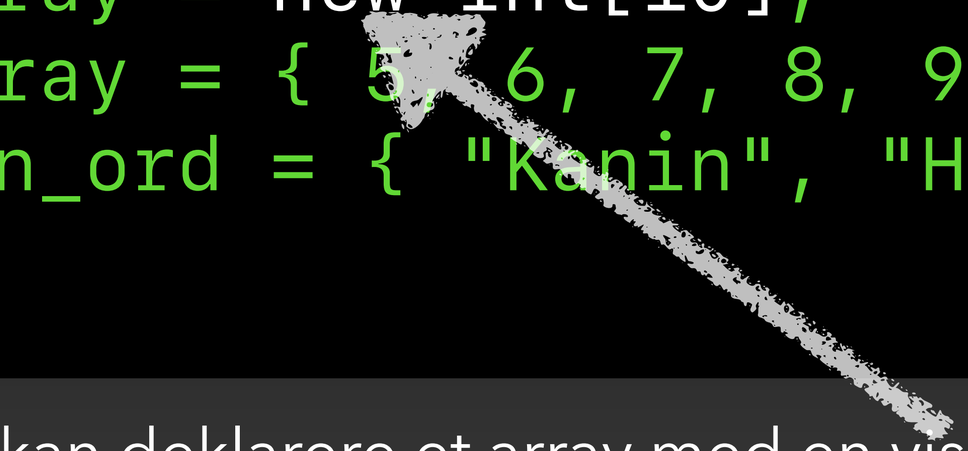
```
int[] fin_array;
int[] min_array = new int[10];
int[] din_array = { 5, 6, 7, 8, 9 };
String[] noen_ord = { "Kanin", "Hund", "Katt" };
```



Vi kan deklarere et array uten å gi den et innhold (uninitialized)

Litt mer om arrays

```
int[] fin_array;
int[] min_array = new int[10];
int[] din_array = { 5, 6, 7, 8, 9 };
String[] noen_ord = { "Kanin", "Hund", "Katt" };
```



Vi kan deklarere et array med en viss størrelse

Elementene vil da få en standardverdi, som for int er 0

Litt mer om arrays

```
int[] fin_array;
int[] min_array = new int[10];
int[] din_array = { 5, 6, 7, 8, 9 };
String[] noen_ord = { "Kanin", "Hund", "Katt" };
```



Vi kan også gi de bestemte verdier

Husk at dette også bestemmer størrelsen for arrayet

Litt mer om arrays

```
int[] min_array = new int[10];
int[] din_array = { 5, 6, 7, 8, 9 };
String[] noen_ord = { "Kanin", "Hund", "Katt" };
min_array[2] = 20;      // 0 -> 20
noen_ord[1] = "Fugl"; // "Hund" -> "Fugl"
```



Vi kan endre en verdi for elementer i et array

Men de må være samme type som den er deklarert som

Litt mer om arrays

```
int[][] to_dimensjoner = {
    { 1, 2, 3, 5, 8 },
    { 3, 6, 8, 9, 2 }
}
```



Arrays kan være i flere dimensjoner

Slik som et 2D array kan være nyttig for et brett i et spill

Litt mer om arrays

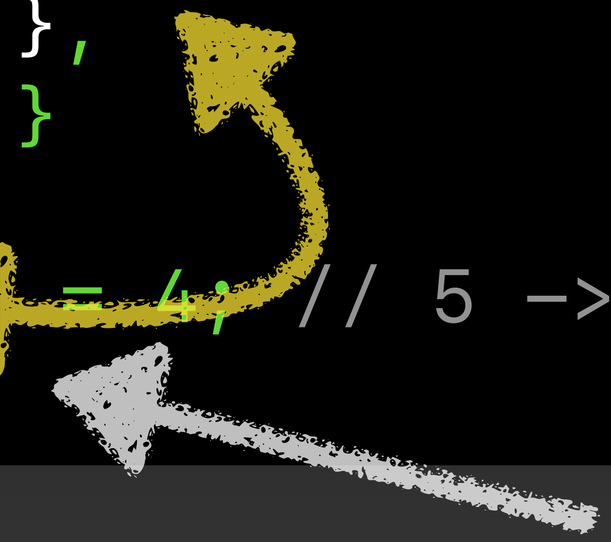
```
int[][] to_dimensjoner = {
    { 1, 2, 3, 5, 8 },
    { 3, 6, 8, 9, 2 }
};
to_dimensjoner[0][3] = 4; // 5 -> 4
```



Vi aksesserer elementer slik, og den ytterste kommer først

Litt mer om arrays

```
int[][] to_dimensjoner = {
    { 1, 2, 3, 5, 8 },
    { 3, 6, 8, 9, 2 }
};
to_dimensjoner[0][3] = 4; // 5 -> 4
```



Vi aksesserer elementer slik, og den ytterste kommer først

Litt mer om arrays

```
int[][] to_dimensjoner = {
    { 1, 2, 3, 5, 8 },
    { 3, 6, 8, 9, 2 }
};
to_dimensjoner[0][3] = 4; // 5 -> 4
```



Vi aksesserer elementer slik, og den ytterste kommer først

Litt mer om arrays

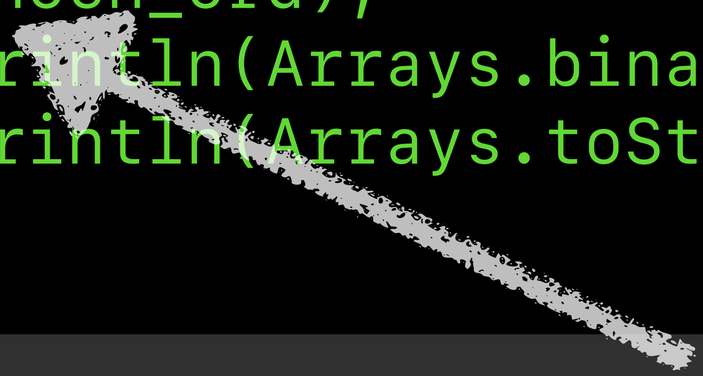
```
import java.util.Arrays;
…
String[] noen_ord = { "Kanin", "Hund", "Katt" };
Arrays.sort(noen_ord);
System.out.println(Arrays.binarySearch(noen_ord, "Hund"));
System.out.println(Arrays.toString(noen_ord));
```

For å manipulere arrays, så har vi en egen klasse til det, som må importeres

Vi kan blant annet søke opp et element, men det må være sortert først

Litt mer om arrays

```
import java.util.Arrays;
…
String[] noen_ord = { "Kanin", "Hund", "Katt" };
Arrays.sort(noen_ord);
System.out.println(Arrays.binarySearch(noen_ord, "Hund"));
System.out.println(Arrays.toString(noen_ord));
```



Så vi kan sortere

I tilfellet her, så vil resultatet bli i det som er *String*-klassens naturlige rekkefølge

Litt mer om arrays

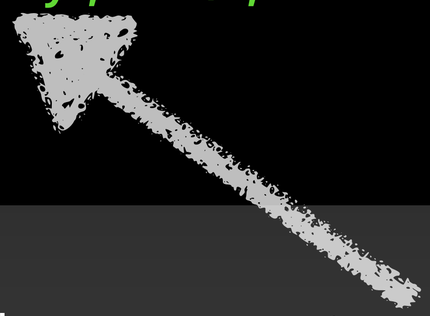
```
import java.util.Arrays;
…
String[] noen_ord = { "Kanin", "Hund", "Katt" };
Arrays.sort(noen_ord);
System.out.println(Arrays.binarySearch(noen_ord, "Hund"));
System.out.println(Arrays.toString(noen_ord));
```



En enkelt måte for å skrive ut alle elementene

Litt mer om arrays

```
import java.util.Arrays;
…
int[] min_array = new int[10];
Arrays.fill(min_array, 5);
```



Gir alle elementene i arrayet en bestemt verdi

Tekst-manipulering

```
String tekst = "Dette er en liten tekst";
char bokstav = 'A';
```

Når vi behandler tekst så bruker vi normalt to datatyper

En for tekst (klassen *String*) og en for enkeltbokstaver (primitiv-typen *char*)

Tekst-manipulering

```
String tekst = "Dette er en liten tekst";
char bokstav = tekst.charAt(7);
int lengde = tekst.length();
String[] ord = tekst.split(' ');
String trimmet = tekst.trim();
```



String-klassen har mange metoder

Vi kan hente ut en enkelt bokstav

Tekst-manipulering

```
String tekst = "Dette er en liten tekst";
char bokstav = tekst.charAt(7);
int lengde = tekst.length();
String[] ord = tekst.split(' ');
String trimmet = tekst.trim();
```



Vi kan finne lengden til teksten

Vær obs på at dette er en metode (parenteser). (arrays bruker instansvariabel)

Tekst-manipulering

```
String tekst = "Dette er en liten tekst";
char bokstav = tekst.charAt(7);
int lengde = tekst.length();
String[] ord = tekst.split(' ');
String trimmet = tekst.trim();
```



Vi kan splitte teksten, i dette tilfellet ved hvert mellomrom

Tekst-manipulering

```
String tekst = "Dette er en liten tekst";
char bokstav = tekst.charAt(7);
int lengde = tekst.length();
String[] ord = tekst.split(' ');
String trimmet = tekst.trim();
```



Trim fjerner mellomrom og annen "whitespace" foran og etter strengen

Tekst-manipulering

```
String et_dyr = "Hest";
String en_blomst = "Rose";
int antall = 2;
String sammen = et_dyr + " og " + en_blomst;
String antall_sammen = antall + " hester";
```

I Java så kan vi sette sammen tekster ved hjelp av pluss

Tekst-manipulering

```
String et_dyr = "Hest";
String en_blomst = "Rose";
String sammen = et_dyr + " og " + en_blomst;
String en_tekst = "Hest og Rose";
if (en_tekst.equals(sammen)) {
    System.out.println("To like tekster");
}
if (en_tekst == sammen)) {
    System.out.println("To like objekter");
}
```

Vi kan sammenligne to tekster, men vær obs på forskjellene her

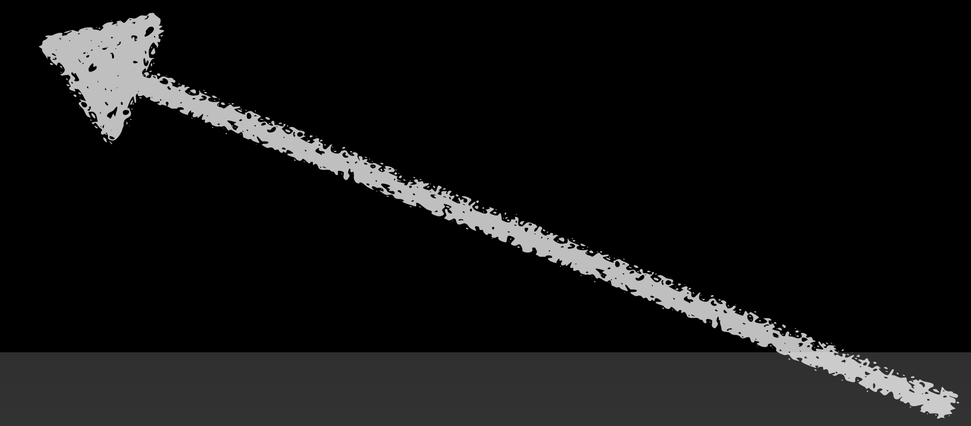
Tekst-manipulering

```
String tallet_to = "2";
int tall = Integer.parseInt(tallet_to);
double flyttall = Double.parseDouble(tallet_to);
```

Når vi har en tekst-streng som er et tall, så kan Java konvertere til en primitiv-type

Løkker

```
int a = 100;
while (a > 0) {
    a--;
}
```



While løkker reperterer så lenge betingelsen er sann

Løkker

```
int a = 100;
do {
    a--;
} while (a > 0)
```



Do/while-løkker kjører en gang, og reperterer deretter så lenge betingelsen gjelder

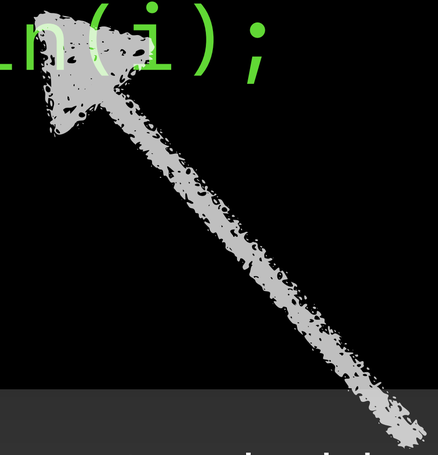
Løkker

Vi har to typer *for*-løkker:

1) **for (initialisering; betingelse; repetisjon)**
   - Initialisering – kjøres en gang
   - Betingelse – repeterer løkken så lenge denne er sann
   - Repetisjon – kjøres etter hver løkke
2) **for (variabel: beholder)**
   - Variabel – definerer type og variabelnavn
   - Beholder – beholderen som skal itereres over

Løkker

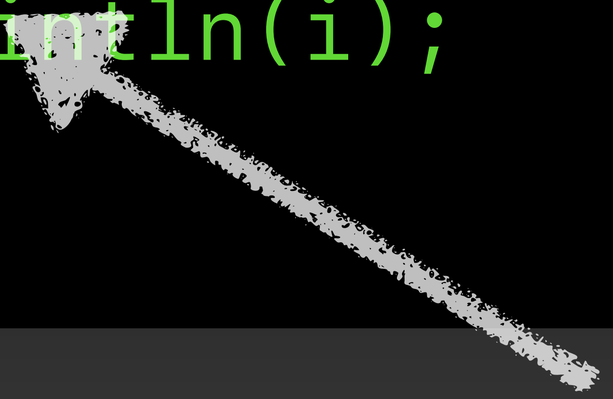
```
for (int i = 100; i >= 0; i--) {
    System.out.println(i);
}
```



Kjører så lenge betingelsen er sann, men siste ledd vil typisk brukes for styring av variabelen. Men kan også skje inne i løkke-blokken.

Løkker

```
int[] fineTall = { 11, 14, 9, 36, 42, 69 };
for (int i: fineTall) {
    System.out.println(i);
}
```



Kjøres for hvert element i beholderen, og hvert element kopieres til variabelen

Løkker

```
import java.util.stream.IntStream;
…
for (int i: IntStream.range(0, 10).toArray()) {
    System.out.println(i);
}
```

Hvis du er glad i Pythons range(), så finnes tilsvarende i Java

Forskjellen mellom a++ og ++a

```
int a = 20;
int b = a++; // b = 20
int c = a;   // c = 21
int d = ++a; // d = 22
int e = a;   // e = 22
```

Har bare betydning ved tilordning

Plussene først gir den nye verdien, plussene etter gir den gamle

null

```
String nullTekst = null;
nullTekst.length();
```

Dette vil ikke kompileres, men gi en så kalt NullPointerException

Det viktigste å vite nå, er at vi bruker *null* kun når vi vil vise at vi ikke har en referanse til et aktuelt objekt.

Vi kan bruke det for å sjekke om et objekt eksisterer (peker på en minneadresse)

Andre datastrukturer

```
import java.util.ArrayList;
…
ArrayList <Integer> tallListe = new ArrayList<>();
tallListe.add(10);
tallListe.get(0);
tallListe.remove(0);
```

ArrayList er en array som har egenskaper som en liste i Python

Størrelsen er dynamisk

Andre datastrukturer

```
import java.util.HashMap;
…
HashMap <String,Integer> ordbok = new HashMap<>();
ordbok.put("Hester", 2);
ordbok.put("Katter", 3);
System.out.println(ordbok.get("Hester"));
```

HashMap fungerer på samme måte som ordbok (dictionary) i Python