Basal programmering

Faglige læringsmål

* At arbejde med simple console-programmer.
* At lære de basale programmeringsinstruktioner i C#
* At få indledende kendskab til programmering som et værktøj **til** at løse konkrete problemer

Konkrete elementer

* De basale programmeringsinstruktioner i C#
  + Udskrivning til konsol, indlæsning fra konsol
  + Variabler, assignment, udtryk
  + If-betingelser
  + Løkkestrukturer (while, do, for, foreach)
  + Metoder (procedurer, funktioner)
* Problemløsning i C# med brug af løkker og betingelser.

Forudsætninger

* Basalt kendskab til Visual Studio, kan oprette projekt/solution, kan starte et program.
* Forståelse for at alle linjer afsluttes med karakteren ’;’.

Indhold

[1. Introduktion 4](#_Toc73095892)

[1.1. Hvad skal vi lære 4](#_Toc73095893)

[1.2. Rammer 4](#_Toc73095894)

[1.3. Skal jeg læse alt det? 4](#_Toc73095895)

[1.4. Skal jeg løse alle opgaver i dokumentet? 5](#_Toc73095896)

[1.5. Oversigter over begreber og basal syntax 5](#_Toc73095897)

[2. Methods grundlæggende 5](#_Toc73095898)

[2.1. Opgave: Læg Hello World i en method 6](#_Toc73095899)

[2.2. Hvor kan man skrive kode? 7](#_Toc73095900)

[2.3. Opgave: Din egen kodelinje 7](#_Toc73095901)

[2.4. Opgave: Din egen metode 7](#_Toc73095902)

[3. Udskrivning og indlæsning 7](#_Toc73095903)

[3.1. Eksempel 1: Write og WriteLine 9](#_Toc73095904)

[3.2. Opgave: Pæn udskrift 9](#_Toc73095905)

[3.3. Eksempel 2: Write og ReadLine kan også kombineres. 9](#_Toc73095906)

[3.4. Opgave: Write/WriteLine 11](#_Toc73095907)

[4. Variabler, typer, assignment, udtryk, konvertering 11](#_Toc73095908)

[4.1. int (integer - det betyder heltal) 11](#_Toc73095909)

[4.2. String (tekst) 12](#_Toc73095910)

[4.3. Andre typer 12](#_Toc73095911)

[4.4. Assignment 12](#_Toc73095912)

[4.4.1. Hvad er et udtryk? 12](#_Toc73095913)

[4.4.2. Udførsel af assignment 13](#_Toc73095914)

[4.4.3. Opgave: Tæl op 13](#_Toc73095915)

[4.4.4. Opgave: Udskriv resultatet af et udtryk 13](#_Toc73095916)

[4.5. Konvertering 14](#_Toc73095917)

[4.5.1. Metoden ToString() 14](#_Toc73095918)

[4.5.2. Konverteringer fra string til andre typer 14](#_Toc73095919)

[4.5.3. Opgave: Convertering af ReadLine 15](#_Toc73095920)

[4.5.4. Konvertering helt generelt 15](#_Toc73095921)

[4.6. Opgave: Navn og hobby 16](#_Toc73095922)

[5. Betingede sætninger 16](#_Toc73095923)

[5.1. if-sætningen 17](#_Toc73095924)

[5.2. else if 17](#_Toc73095925)

[5.3. Simple betingelser 18](#_Toc73095926)

[5.4. Sammensætning af betingelser 19](#_Toc73095927)

[5.5. Opgave: Betingelser 19](#_Toc73095928)

[5.6. switch – den venter vi med lige nu 19](#_Toc73095929)

[5.7. Opgave: BMI 20](#_Toc73095930)

[6. Løkkestrukturer 20](#_Toc73095931)

[6.1. while løkker 20](#_Toc73095932)

[6.2. Opgave: Simpel while loop 21](#_Toc73095933)

[6.3. Opgave: Main loop 21](#_Toc73095934)

[6.4. Do while 21](#_Toc73095935)

[6.5. Opgave: Kommandohåndtering 22](#_Toc73095936)

[6.6. for løkker 23](#_Toc73095937)

[6.7. Opgave: Jeg må ikke stjæle 24](#_Toc73095938)

[6.8. Opgave: Fødselsdag (lidt kringlet at få til at fungere helt rigtigt) 24](#_Toc73095939)

[6.9. Opgave: Division 24](#_Toc73095940)

[6.10. foreach løkker 25](#_Toc73095941)

[7. Arrays 25](#_Toc73095942)

[7.1. Array kort fortalt 25](#_Toc73095943)

[7.2. Oprettelse af arrays 25](#_Toc73095944)

[7.3. Brug af arrays 26](#_Toc73095945)

[7.4. Opgave: Lidt array-gymnastik 27](#_Toc73095946)

[8. Løkker og arrays 27](#_Toc73095947)

[8.1. Opgave: Spillernavne 27](#_Toc73095948)

[8.2. Hvordan man løser opgaver med løkker - brug en kandidat 27](#_Toc73095949)

[8.3. Opgave: Bowling 28](#_Toc73095950)

[8.4. Ekstraopgave: Bowling 28](#_Toc73095951)

[8.5. foreach-løkker 28](#_Toc73095952)

[8.6. Opgave: Udskriv alle navne med foreach 29](#_Toc73095953)

[9. Methods udvidet 29](#_Toc73095954)

[9.1. Parametre 29](#_Toc73095955)

[9.2. Opgave: HelloMan udvidelse 30](#_Toc73095956)

[9.3. Opgave: method parametre (BMI) 30](#_Toc73095957)

[9.4. Flere parametre (BMI) 30](#_Toc73095958)

[9.5. Returværdi 30](#_Toc73095959)

[9.6. Opgave: BMI funktion 31](#_Toc73095960)

[9.7. Opgave: Rest ved division 31](#_Toc73095961)

# Introduktion

For at kunne komme i gang med programmering, skal man kende de forskellige instruktioner C# forstår. I dette dokument skal vi arbejde med de mest basale instruktioner i programmering, instruktioner der findes i alle almindelige programmeringssprog (imperative sprog).

## Hvad skal vi lære

* Vi skal lære at gemme værdier (f.eks. tal eller tekster) i variabler og ændre i dem.
* Vi skal lære at lade programmet udføre forskellige handlinger baseret på om en betingelse er sand eller falsk.
* Vi skal lære at lave løkker der kan få programmet til at foretage samme operation flere gange indtil den samlede opgave er løst.

Dokumentet her er opbygget sådan at der skiftevis er forklaringer C# instruktioner og opgaver i at bruge dem. Hvis man tænker man godt ved det der står i forklaringerne, kan man springe direkte til opgaverne – kan man løse opgaverne, har man også forstået forklaringerne.

I princippet er der tale om forskellige opgaver som kunne løses ved at oprette et nyt projekt hver gang. Det anbefales dog at løse alle opgaverne her i det samme projekt sådan at man ikke får opbygget en stor mængde af små ubrugelige projekter – det vil være nemmere at have ét projekt man skal kigge i hvis man lige skal undersøge hvordan det nu var man f.eks. lavede en betingelse eller en løkke.

## Rammer

I dette dokument om basal programmering benyttes rammer for at fremhæve nogle dele, oftest eksempler. Der benyttes rammer i to farver, grøn og rød. Den grønne ramme bruges til at vise C#-kode, den røde til at vise hvad der kommer ud på skærmen.

Kode i grøn ramme:

// Dette er en C# kommentar

Kode i rød ramme:

Dette er hvad der sker på skærmen.

## Skal jeg læse alt det?

Der er en del tekst her i dokumentet. Hensigten er at man skal kunne finde alle de oplysninger man har brug for. Nogle gange er meget af teksten sikkert overflødig fordi du godt ved det i forvejen, eller fordi du godt kan gennemskue det ud fra eksemplerne.

Hvis du umiddelbart kan forstå eksemplerne og løse opgaverne, er der ikke nogen grund til at læse teksten, så forstår du det godt.

Hvis du undervejs kører lidt træt i denne vejledning, findes der også en del online-materiale om programmering. Det kan nogle gange være en hjælp at følge en online tutorial. Snak med din underviser om det kunne være bedre for dig at prøve noget online materiale. Onlinemateriale kan du selv søge på nettet, ellers er der nogle eksempler på materiale her:

[Eksempler på online-materiale om grundlæggende programmering.](https://drive.google.com/open?id=1pc7LjyjQsP1r6hUHVw7kZambUXFYOOai)

## Skal jeg løse alle opgaver i dokumentet?

Som udgangspunkt: Ja. Men spørg din lærer, måske kan det give god mening at springe en enkelt opgave over.

Hvis du tænker opgaverne er meget nemme, kan det godt være du lige skal spørge din underviser om ikke du skal have noget der er sværere. På den anden side, hvis opgaven er nem, så kan du også hurtigt løse den, og måske var der noget i opgaven du ikke tænkte på før du kom i gang med at løse den.

Hvis opgaven er svær, så få hjælp. Spørg din lærer eller en af de andre elever.

## Oversigter over begreber og basal syntax

Vi kommer ikke omkring alt i C# i dette dokument, kun det basale for at komme i gang med at programmere, de emner vi mangler vil vi lære efterhånden som vi får behov for det.

Jeg har lavet en oversigt over forskellige begreber i C# og samlet dem i et dokument. Hvis du støder på begreber der forvirrer dig, kan du måske finde en kort beskrivelse ved at følge dette [link til C-sharp begreber](https://intranet.aspit.dk/fagomrader/softwarekonstruktion/Documents/Materialer/CLSC/Teoritekster/Generelt/C-sharp%20begreber.docx?Web=1).

Når man skal skrive noget C#-syntax kan man ikke altid huske hvad der skal skrives hvor. I dokumentet [Erklæringer i C-sharp](https://drive.google.com/open?id=1iXW8fN6KYML_8YVPosyAzmtforfq86W8) kan man finde præcise angivelser af hvordan man bruger de mest almindelige C# konstruktioner.

# Methods grundlæggende

Methods er et ret omfattende begreb som vi ikke skal lære alt om endnu. Men da vi allerede har lavet vores første method bare ved at oprette et projekt og i opgaverne nedenfor kommer til at lave flere, bliver vi nødt til at vide lidt om hvad det er for noget. Methods er som navnet antyder en metode til at gøre et eller andet (i andre sprog kaldes de også en procedure eller en funktion). Det vil sige at hver gang man vil have gjort et eller andet, så kan man *kalde* denne method i stedet for at skrive noget kode der gør det samme. Det er med andre ord en måde at undgå at skulle skrive det samme kode igen og igen. Hvis du har hørt begrebet procedurel programmering, så henviser betegnelsen til det at benytte procedurer (som i C# kaldes methods) til at håndtere grupper af kode man gerne vil have udført flere gange.

Som nævnt, har vi allerede lavet én method, det er den der hedder Main. Metoden Main kaldes automatisk når programmet startes.

static void Main(string[] args)

{

Console.WriteLine("Hello world");

}

Da det er en console application er metoden static, hvilket fremgår af ordet static i koden. De metoder du opretter i forbindelse med opgaverne i dette dokument skal også være static.

Ordet void betyder at metoden ikke sender information tilbage. Med mindre du får andet at vide, skal dine metoder også være void i opgaverne i dette dokument.

Efter metodens navn Main står der (string[] args). Det der står inden i parentesen er det information der sendes til metoden fra den der kalder metoden. Det ser lidt kryptisk ud, du skal ikke bruge det til noget. Indtil videre har vi ikke brug for at sende information til de metoder vi selv skal lave, så indtil videre skriver bare "()" efter vores egne metodenavne. Parenteserne er vigtige, det er dem der fortæller at der er tale om en metode.

Lad os nu lave vores første metode:

static void Hello()

{

Console.WriteLine("Hello world");

}

Metoden Hello implementeres på samme niveau i klassen Program som Main(), f.eks. neden under Main-metoden.

Så nu har vi flyttet linjen der udskriver teksten "Hello world" i en metode, så kan vi fjerne den gamle kode fra Main og i stedet *kalde* (bede den om at udføre) metoden Hello().

static void Main(string[] args)

{

Hello();

}

I dette tilfælde er der ikke vundet så meget, vi har mest gjort det for at se hvad en metode er. Når du løser opgaverne i dette dokument, så løs hver opgave i sin egen metode. Så har du alle opgaverne i samme projekt, og du kan rette Main til sådan at den *kalder* den eller de metoder du gerne vil have udført.

Det endelige program kommer til at se sådan ud:

class Program

{

static void Main(string[] args)

{

Hello();

}

static void Hello()

{

Console.WriteLine("Hello world.");

}

}

Vi går mere i dybden med methods i afsnit *9 Methods udvidet*.

## Opgave: Læg Hello World i en method

Læg Hello World koden ind en metode som vist ovenfor.

## Hvor kan man skrive kode?

Ja, det spørgsmål er lidt mærkeligt, for hvis du ser i en fil, så er der kode over det hele. Det der menes med spørgsmålet er hvor man kan skrive *instruktioner* der beder computeren om at udføre noget. Hvis vi ser eksemplet ovenfor, så er der to linjer hvor computeren faktisk bliver bedt om at udføre noget, nemlig linjen i ”kroppen” af metoden Main og linjen i kroppen af metoden Hello.

## Opgave: Din egen kodelinje

Tilføj en linje kode til programmet (et sted hvor man kan skrive instruktioner) der udskriver en tekst (f.eks. ”Simsalabim”). Kør programmet, test at ordet udskrives.

## Opgave: Din egen metode

Tilføj en metode der hedder Trylleri. Metoden skal udskrive teksten ”hokus pokus”.

# Udskrivning og indlæsning

Når man kører et program, så foregår der en masse ting som man ikke nødvendigvis kan se. Den der bruger programmet kan kun se hvad programmet skriver ud på skærmen. Så hvis programmet ikke skriver noget (eller på en anden måde kommunikerer med omverdenen) er der ikke meget opnået ved at køre programmet.

Nedenfor er de mest almindelige metoder til at udskrive og indlæse fra konsolvinduet.

#### Udskrive:

Console.Write("tekst"); // Udskriver

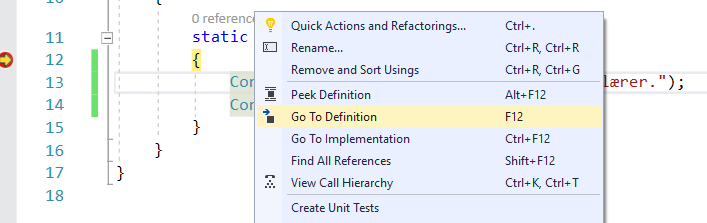
Console.WriteLine();    // Udskriver og skifter linje

#### Indlæse:

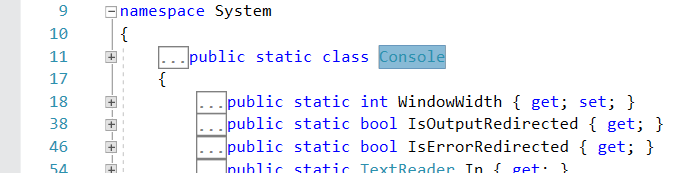
Console.ReadLine();   // Indlæser indtil linjeskift

Console henviser til den klasse metoderne ligger i, mens Write, WriteLine og Readline er de konkrete metodenavne.

Prøv at højreklikke på Console og vælg Go To Definition:



Nu åbner Visual Studio klassen Console:



Den består af mange metoder, Write, WriteLine og ReadLine findes et stykke nede. Implementationen af Console-biblioteket er skjult for brugeren, man kan kun se metodernes signatur, ikke hvordan de er implementeret.

Man skal altså altid skrive ”Console.” Før alle instruktionerne Write, WriteLine, ReadLine etc. (også selvom det skulle være glemt i opgaven ;-)

## Eksempel 1: Write og WriteLine

Her er et eksempel på brugen af Write og WriteLine som illustrerer hvordan man kan benytte dem til at gøre sin udskrift mere læsbar.

Console.Write("Hans");

Console.Write("og");

Console.Write("Grete");

Console.Write("for vild i skoven.");

Console.Write("Den næste dag gik de videre.");

Console.ReadLine();

Giver følgende udskrift i konsollen (i konsolvinduet på skærmen):

HansogGretefor vild i skoven.Den næste dag gik de videre.

Det er jo ikke særlig nemt at læse. Der mangler mellemrum mellem nogle af ordene og det hele står på én linje. Når man benytter flere Write og WriteLine-kald efter hinanden, skal man huske selv at sørge for mellemrum og linjeskift de rigtige steder. Vi retter koden til

Console.Write("Hans");

Console.Write(" og");

Console.Write(" Grete");

Console.WriteLine(" for vild i skoven.");

Console.WriteLine("Den næste dag gik de videre.");

Console.ReadLine();

Nu får vi denne udskrift:

Hans og Grete for vild i skoven.

Den næste dag gik de videre.

Det ser bedre ud.

## Opgave: Pæn udskrift

Du skal nu lave et program der består af mindst tre kald af Write eller WriteLine. Programmet skal skrive følgende ud på skærmen.

Dig og mig og vi to sejled i en træsko

Da vi kom til Langeland var vor træsko fuld af vand

## Eksempel 2: Write og ReadLine kan også kombineres.

Betragt følgende gode program:

Console.Write("Hvem er den bedste lærer? ");

string lærer = Console.ReadLine();

Console.WriteLine(lærer **+** " er den bedste lærer.");

Console.ReadLine();

Da den sidste Write-instruktion ikke indeholder et linjeskift, bliver brugeren bedt om at indtaste svaret på samme linje i det efterfølgende kald af Console.ReadLine(). Metoden Console.ReadLine *returnerer[[1]](#footnote-1)* en string, nemlig den som brugeren indtaster. Ved at gemme returværdien i en variabel, kan vi fastholde det som brugeren indtaster.

Ovenstående program giver følgende skærmdialog:

Hvem er den bedste lærer? Claus

Claus er den bedste lærer.

Brugerens indtastning

Nu har vi sæt at vi kan sætte strenge (tekster) sammen ved hjælp af operatoren ’+’. Det kan være en ret nem måde at håndtere strenge.

Der findes også en anden ret smart måde. Hvis man skriver symbolet ’$’ foran strengens begyndelse, altså lige foran anførselstegnet, fortæller man compileren at det der står i {} skal beregnes som et udtryk.

Eksempel:

Console.Write("Hvem er den bedste lærer? ");

string lærer = Console.ReadLine();

Console.WriteLine($"**{lærer}** er den bedste lærer.");

Console.ReadLine();

Dette giver samme dialog som vist ovenfor.

Skriv ’$’ foran første anførselstegn.

Indsæt det beregnede udtryk i ”krøllet parentes”.

I eksemplet ovenfor er det beregnede udtryk bare en variabel, men det kan være hvilken som helst værdi der kan fremkomme ved en beregning. Eksempel:

Console.WriteLine($"Gennemsnit: **{sum / Max(antal, 1)}**");

I dette eksempel indgår både et metodekald og beregning af division.

## Opgave: Write/WriteLine

Skriv en metode der udskriver nedenstående tekst. Du skal mindst benytte *et* kald af metoden Write(). Ellers må du bruge alle de Write- og WriteLine-kald du vil.

Brugeren bliver først bedt om at indtaste fire ord, et navneord, et udsagnsord, et tillægsord og så endnu et navneord. Ordene som *brugeren indtaster* skal programmet sætte ind i et lille digt (Kim Larsen sang). Hvis brugeren taster de rigtige ord, får man det originale digt, ellers får man noget andet.

Indtast navneord: succes

Indtast udsagnsord i nutid: hører

Indtast tillægsord i flertal (f.eks. smukke): fede

Indtast navneord: musik

Masser af succes

Og det der hører til

Masser af fede spotlights

Privatliv og musik

Brugerens indtastning efter ’:’

Det er vigtigt i denne opgave at brugerdialogen ser ud på samme måde som ovenfor, altså at programmet starter med indsamling af al information, hvorefter det genererede digt udskrives.

# Variabler, typer, assignment, udtryk, konvertering

C# er et stærkt typet (strongly typed) sprog. Det betyder at alle de variabler man vil bruge skal erklæres, det vil sige at man skal skrive i koden hvilken type de har og variablens navn. F.eks.:

int count;

## int (integer - det betyder heltal)

Ovenstående linje kode erklærer (opetter) en variabel der hedder count, variablen er af typen int, hvilket er kort for integer som betyder heltal. Variablen count kan altså have alle heltal[[2]](#footnote-2) som værdier.

Variabler af typen int benyttes i meget stort omfang i programmering.

Samtidig med at man erklærer en variabel, kan man give den en startværdi (initiel værdi). Nedenstående linje erklærer heltalsvariablen count og giver den værdien 0:

int count = 0;

## String (tekst)

Typen string (på dansk “streng”) bruges til variabler der skal indeholde tekst. Denne type benyttes også i meget høj grad.

string name;

string city = "Aarhus";

Ovenstående linjer erklærer to string-variabler name og city. Variablen city får startværdien ”Aarhus”. Strengværdier angives altid i anførselstegn, sådan at compileren (det program der oversætter C#-koden til kode computeren kan udføre) kan se forskel på en værdi og en variabel.

## Andre typer

Der er flere andre datatyper, nedenfor nævnes nogle af de vigtigste. Vi kommer til at beskæftige

uint (unsigned int – positive heltal)

bool (boolean – sandhedsudtryk – to værdier true og false)

char (character – et enkelt tegn)

float, double (flydende kommatal, double er mere præcis end float)

decimal (decimaltal, mere præcis end double, bruges til beløb)

## Assignment

Vi har allerede set eksempler på *assignment* ovenfor. *Assignment* er det der sker når variablen given en værdi. Vi har givet variablen count værdien 0 og variablen city værdien "Aarhus" i rammerne ovenfor. I de linjer udnyttede vi at man kan lave erklæring og et initialiserende *assignment* i samme linje.

En *assignment*-instruktion består af en variabel, et lighedstegn og et udtryk:

<variabel> = <udtryk>

### Hvad er et udtryk?

Et udtryk er en eller anden form for beregning der resulterer i en værdi af *samme type* som variablen. Hvis variablen er af typen int, skal udtrykket give en værdi af typen int. Følgende er altså ikke tilladt:

count = city;

Til gengæld er følgende linje kode helt gyldig:

count = city.Length;

Hvis værdien af city er "Aarhus" vil resultatet af linjen være at variablen count får værdien 6 som er længden af city’s værdi (altså længden af strengen "Aarhus").

Udtryk kan være vilkårligt komplekse. For eksempel beregner følgende udtryk den gennemsnitlige længde af to bynavne:

(city1.Length + city2.Length) / 2

Dette udtryk kan f.eks. bruges i en assignment-instruktion (se afsnit 4.4.2).

Gennemsnit = (city1.Length + city2.Length) / 2

Udtryk kan også være af andre typer:

city1 + "-" + city2

Hvis city1 og city2 er strenge og lig henholdsvis "Aarhus" og "Odder", er resultatet af ovenstående udtryk strengen "Aarhus-Odder", altså en sammensætning af de tre strenge[[3]](#footnote-3) - en med et engelsk ord såkaldt concatenation.

### Udførsel af assignment

Når et assignment afvikles beregnes udtrykket først, hvorefter værdien af udtrykket gemmes i variablen.

snit = (city1.Length + city2.Length)/2;

sammensat = city1 + city2;

Overvej hvad følgende linje kode gør:

count = count + 1;[[4]](#footnote-4)

Der er altså forskel på et matematisk lighedstegn og et assignment lighedstegn i C#. Et matematisk lighedstegn skrives i C# med to lighedstegn, altså ==. Det kommer vi nærmere ind på i afsnit 5 nedenfor.

### Opgave: Tæl op

Lav en lille metode der beder brugeren om at indtaste et tal. Tallet skal gemmes i en variabel i programmet. Derefter skal variablen sættes til en værdi der er 12 større. Til sidst skal variablen udskrives.

Det kunne give en dialog som ser omtrent således ud:

Indtast et tal: 4

Variablen er nu øget med 12 og har fået værdien **16**.

Brugerens indtastning.

### Opgave: Udskriv resultatet af et udtryk

Et taxaselskab tilbyder at man kan beregne prisen på forhånd. Taxaselskabet opgiver sine priser således:

På hverdagskørsel fra kl. 04 til kl. 20 opkræves en starttakst på 29 kr. og en kilometertakst på 9 kr.

Du skal lave et lille program som beregner prisen for en taxatur. Dialogen kunne se således ud:

>>Velkommen til taxaberegner<<

Indtast antal kilometer: 24

Pris for taxatur: 245

Brugerens indtastning.

## Konvertering

Nogle gange kan man have behov for at konvertere variabler af forskellig type, f.eks. lave en streng om til et tal (hvis strengen har en værdi der giver mening at opfatte som et tal). Det modsatte også ret ofte et ønske.

### Metoden ToString()

Alle variabler og objekter (en variabel sammensat af flere ting) har en metode der hedder ToString(). På simple datatyper som f.eks. int vil resultatet af konverteringen være den værdi man forventer:

int age = 51;

string ageString = age.**ToString()**;

Console.WriteLine($"ageString er: {ageString}");

Giver følgende resultat:

ageString er: 51

### Konverteringer fra string til andre typer

Når man skal konvertere en værdi fra string til en anden type kan man benytte metoden Parse eller TryParse hvis man vil være sikker på programmet ikke crasher. Den metode er defineret på alle standardtyper.

int age;

decimal price;

age = **int.Parse**("33");                     // Do like this

age = int.Parse(Console.ReadLine());       // Do like this

age = int.Parse("33 år");                  // Error

double height;

height = 1.78;                 // Use "." In C#

height = **double.Parse**("1,78"); // Use "," outside C# (Danish komma)

height = double.Parse(Console.ReadLine()); // Parse input

// TryParse - no crash

int.TryParse("33", out age);

decimal.TryParse("331,95", out price);

Console.WriteLine($"age: {age}, height: {height}, price: {price}");

Linjen 'age = int.Parse("33");' læser strengen 33 igennem og konverterer den til en int (med værdien 33).

Linjen 'age = int.Parse(Console.Readline());' læser den tekst brugeren taster ind under afvikling af programmet og konverterer den til en int. Denne linje kode vil man ofte se i simple console programmer.

Den næste linje 'age = int.Parse("33 år");' forsøger samme konvertering men på strengen "33 år". Men det kan ikke lade sig gøre, for strengen er ikke et tal. Derfor crasher programmet[[5]](#footnote-5).

Hvis man vil være sikker på at konverteringen ikke får programmet til at crashe, kan man benytte metoden int.TryParse. Den *metode* returnerer en boolean og kan derfor bruges sammen med betingede sætninger (se afsnit 5 Betingede sætninger). Linjen 'int.TryParse("33", out age); ' læser teksten "33" igennem, forsøger at læse tallet som et heltal og hvis det lykkes sættes værdien ind i variablen age. Det reserverede ord out bruges når *metoder* skal kunne ændre i værdien af *variabler* der gives med ind i *metoden*. Det er ikke så vigtigt at forstå lige nu, du skal bare vide at dette er en måde at konvertere strengværdier til andre typer – man skal huske at skrive out ved den variabel resultatet skal komme ud i.

### Opgave: Convertering af ReadLine

Lav et program som beder brugeren indtaste to tal og som udskriver summen af de to tal., f.eks.

Indtast tal: 6

Indtast tal: 8

Summen af 6 og 8 er 14

### Konvertering helt generelt

Ud over disse konverteringer fra og til string, findes der en større samling konverteringsmetoder i namespacet Convert. Hvis man vil konvertere noget til int, bruges f.eks. Convert.ToInt32 af den værdi man vil konvertere. Der findes 19 ”forskellige” metoder der alle hedder Convert.ToInt32[[6]](#footnote-6) sådan at alle andre indbyggede typer umiddelbart kan konverteres til int.

Der findes tilsvarende metoder til alle indbyggede typer.

Eksempler:

double age = 3.8;

int integerAge;

integerAge = **Convert.ToInt32(age)**;

Console.WriteLine($"Age: {age}, Integer-age: {integerAge}");

double age2 = **Convert.ToDouble(integerAge)**;

Ovenstående kode giver følgende resultat. Bemærk: C# har ikke noget imod at konvertere et decimaltal til et heltal, den runder tallet af til nærmeste heltal.

Age: 3,8, Integer-age: 4

Age2: 4

## Opgave: Navn og hobby

Lav en metode hvor brugeren bliver bedt om at indtaste navn, alder, hobby og hvor længe man har haft den hobby. Hvis man indtaster *Claus, 51,* computerspil og 28, skal programmet udregne hvilket årstal brugeren begyndte at dyrke sin hobby og hvilken alder han/hun havde på det tidspunkt[[7]](#footnote-7):

Hej Claus, din hobby er computerspil. Du begyndte at dyrke computerspil i 1990 i en alder af 23 år.[[8]](#footnote-8)

Bemærk: Denne opgave kræver at programmet udregner tallene i udskriften - det er ikke de samme tal som dem man taster ind. Det er muligt at få computeren til at fortælle hvilket år vi har lige nu, men det vil være fint bare skrive det nuværende årstal direkte i koden (man får brug for det til at beregne et af de to tal i udskriften ovenfor).

# Betingede sætninger

Betingelser er nøglen til at kunne lave programmer der kan gøre andet og mere end at udgøre simple trivielle opgaver. Med betingelser kan vi lave programmer der gør én ting hvis en betingelse er opfyldt og en anden hvis betingelsen ikke er opfyldt.

Hvis man skulle lave programmet i en hæveautomat, kunne det være en betingelse for at udbetale pengene at der stod tilstrækkelig med penge på kontoen.

## if-sætningen

Den mest almindelige C#-konstruktion til betingelser er if-sætningen. Eksempel:

if (saldo > 0)

{

udbetaling = beløb;

}

else

{

udbetaling = 0;

}

Koden viser at hvis der er en positiv saldo (penge på kontoen), så sættes udbetalingsbeløbet til det ønskede beløb og ellers sættes udbetaling til 0, dvs. der udbetales ikke penge fra automaten.

Generelt ser if-sætningen sådan ud i C#:

if (<betingelse>)

{

<kode>

}

Og med else-del:

if (<betingelse>)

{

<instruktioner>

}

else

{

<instruktioner>

}

En if-sætning består altså af det reserverede ord if efterfulgt af en betingelse i parentes, igen efterfulgt af en krop rammet ind af krøllede parenteser.

Ordene indrammet med <> henviser til at her indsættes den konkrete C#-kode der skal bruges i det konkrete tilfælde, så der kan være tale om alle mulige betingelser og instruktionslinjer i kodeeksemplet. Der er uendelig mange forskellige betingelser og uendelig mange muligheder for at skrive kode i kroppen af betingelsen.

## else if

Ofte har man brug for at teste flere forskellige betingelser og alt efter hvilke betingelser der er sande udføre forskellig kode.

Et eksempel kunne være at man ville undersøge om saldoen på en bankkonto var positiv (større end nul), nul eller negativ. Her er der tre svarmuligheder, så vi kan ikke bare bruge else ligesom ovenfor. Ved at benytte instruktionen else if(<condition>) {…}[[9]](#footnote-9), kan man angive at hvis den oprindelige betingelse ikke var sand, så skal en ny betingelse undersøges.

if (amount > 10000)

{

Console.WriteLine("Du er rig");

}

else if (amount > 0)

{

Console.WriteLine("Der er en positiv saldo.");

}

else if (amount == 0)

{

Console.WriteLine("Saldoen er nul.");

}

else

{

Console.WriteLine("Der er en negativ saldo.");

}

Bemærk: Der kan være mange else if betingelser i ”samme” if-konstruktion.

## Simple betingelser

Simple betingelser er en sammenligning af to udtryk. Sammenligninger kan gøres med følgende operatorer (sammenligningsoperatorer):

< mindre end a<b tester om variablen a er mindre end variablen b

<= mindre end eller lig a<=b tester om a er mindre end eller lig b

== lig med a==b *tester* om a er lig b – bemærk forskellen fra a=b som betyder *sæt* a lig b)

!= forskellig fra, eller ikke lig med – ’!’ betyder *not*

> større end a>b tester om a er større end b

>= større end eller lig a>=b tester om a er større end eller lig b)

## Sammensætning af betingelser

Ofte vil man gerne undersøge flere ting i en betingelse, f.eks. vil vi måske udbetale penge i pengeautomaten enten hvis der er penge på kontoen eller hvis kunden er eksklusivkunde. Med andre ord, vi vil gerne kunne sætte betingelser sammen med brug af ’eller’ og ’og’. Det kan vi heldigvis også.

&& og a<b && b<c tester om begge udtryk er sande

|| eller a<b || b<c tester om bare ét af udtrykkene er sande

Bemærk at ordet udtryk kan dække over alle typer i C#, ofte er det tal, men det behøver det ikke være. I ovenstående tilfælde er der tale om boolske udtryk.

Eksempel:

bool eksklusivKunde;

… // (En masse) kode

if (saldo > 0 || eksklusivKunde)

{

udbetaling = beløb;

}

else

{

udbetaling = 0;

}

Bemærk: den anden halvdel af den sammensatte betingelse består kun af en variabel – det ser ud som om der mangler en sammenligningsoperator. Men hvis man ser på første linje i boksen, så kan man se at variablen eksklusiv er af typen bool, det vil sige at dens værdi i sig selv er en sandhedsværdi. Derfor behøver vi ikke sammenligne den med noget.

Som i matematik kan man benytte parenteser til at styre i hvilken rækkefølge udtryk skal beregnes, altså også boolske udtryk.

## Opgave: Betingelser

Lav et lille program hvor brugeren bliver bedt om at indtaste skostørrelse. Derefter udskrives en af følgende tekster:

* Hvis størrelsen er større en 45 -> "Store plader"
* Hvis størrelsen er mellem 38 og 45 -> "Gennemsnitlige fødder"
* Hvis størrelsen er under 38 -> "Små fødder"

## switch – den venter vi med lige nu

Måske har du hørt ordet switch eller case nævnt i forbindelse med betingelser. En switch kan gøre bestemte typer betingelser nemme at skrive og læse, men de kan kun bruges når der sammenlignes med konstante værdier, så man kan ikke bruge komplekse betingelser som ved if-sætninger. Der er nok af nye instruktioner, og switch giver os ikke noget vi ikke kan gøre med if og else if, så vi venter med at introducere den.

## Opgave: BMI

Lav en metode hvor brugeren kan indtaste højde, vægt og køn. Programmet skal beregne BMI og ud fra dette svare på om man er undervægtig, normalvægtig eller overvægtig.

Følg eksempelvis dette link for at finde ud af hvordan BMI beregnes og hvilke BMI-tal der svarer til de tre forskellige kategorier for de to køn: [BMI – Hvordan udregner man sit BMI? – Helsam](https://www.helsam.dk/artikler/slank/bmi)

Indtast din højde i meter: 1,78

Indtast din vægt i kilo: 74

Indtast køn M/K: M

BMI: 23,355636914530993561419012751

Normalvægtig mand

# Løkkestrukturer

Man har ofte behov for at få sit program til at gøre noget gentagne gange. Til dette formål kan man lave løkker der udfører samme stykke kode så længe en betingelse er opfyldt.

## while løkker

while-løkker er den mest generelle løkkestruktur i næsten alle programmeringssprog, også i C#. I princippet er denne struktur den eneste man har brug for, for at kunne løse alle problemer med gentagne hændelser. De andre løkkestrukturer nedenfor er taget med i sproget for at gøre det nemmer at lave løkker der opfører sig på en speciel måde.

Syntaksen for while ligner meget syntaksen for den *simple* if-sætning (dvs. uden else og else if):

while (<betingelse>) {<kode (krop)>}

Forskellen på while og if er: while udfører kodelinjerne igen og igen så længe betingelsen er opfyldt, if udfører kun kodelinjerne én gang.

Eksempel: Vi vil gerne tælle hvor mange gange man kan trækket et tal fra et andet. Man kan f.eks. trække 3 fra 12 fire gange. Hvis man skulle lave et program der kunne gøre dette generelt, kunne det laves med en løkke som vist nedenfor:

int tal = 12; // hardcoded for simplicity

int div = 3; // hardcoded for simplicity

Betingelse

int rest = tal;

int count = 0;

while (rest >= div)

{

rest = rest - div;

Krop

count++; // same as count = count +1;

}

Variablen rest holder øje med hvor meget der er tilbage.

Instruktionen **count++** betyder at variablen count gøres én større, vi siger at den tælles én op. Variablen count tælles altså op hver gang vi har trukket div fra rest. Det vil altså sige den tæller hvor mange gange vi har trukket div fra rest.

Betingelsen (rest >= div) fortæller programmet om det er muligt at trække div fra rest. Så længe det er muligt, fortsætter løkken.

## Opgave: Simpel while loop

Lav en metode med en while-løkke der udskriver teksten "Jeg må ikke lyve" 6 gange.

## Opgave: Main loop

*Lav din Main-metode om* sådan at den bliver ved med at køre i ring. På denne måde standser programmet *aldrig*, så du kan teste det igen og igen uden at skulle starte det op forfra.

Benyt en while løkke til opgaven. Det betyder at du skal angive en betingelse og en krop.

* Da løkken aldrig skal stoppe, skal vi finde en betingelse der altid er sand. Hvilken betingelse (der er flere) er altid sand?
* Kroppen skal være alt det du i forvejen har stående i din Main metode, sådan at det hele bliver udført igen og igen.

Bemærk: I denne opgave skal du *ikke* oprette en ny metode, du skal *ændre* i den eksisterende Main-metode.

## Do while

Nogle gange har man brug for løkker hvor betingelsen først skal tjekkes *efter* første gennemkørsel af løkken. Det er tilfældet når man under alle omstændigheder vil have udført koden mindst én gang.

Så kan man bruge en do-while-konstruktion.

do {<kode>} while (<betingelse>}

Et godt eksempel er når man gerne vil have brugeren til at indtaste noget information for hver gennemkørsel og programmet så skal gå ud af løkken når brugeren skriver noget bestemt.

Eksemplet nedenfor benytter klassen Random. Det er ikke så vigtigt for dette eksempel at du forstår hvordan man bruger klassen, det er nok at vide at linje to giver variablen tal en tilfældig heltalsværdi mellem 0 og 100 (begge inklusiv)[[10]](#footnote-10). Eksemplets formål er at illustrere løkkestrukturen.

Eksempel:

Random rnd = new Random();

int tal = rnd.Next(1, 101);

int guess;

Console.WriteLine("Gæt et tal mellem 1 og 100");

do

{

Console.Write("Dit gæt: ");

guess = Convert.ToInt32(Console.ReadLine());

if (guess < tal)

{

Krop

Console.WriteLine("Dit gæt " + guess + " er for lavt");

}

if (guess > tal)

Betingelse

{

Console.WriteLine("Dit gæt " + guess + " er for højt");

}

} while (guess != tal);

Console.WriteLine("Tillykke, du har gættet tallet " + tal);

Programmet indeholder en løkke der starter med at lade brugeren gætte på det hemmelige tal og derefter fortsætter med at lade brugeren gætte indtil han har gættet det rigtige. Da brugeren altid skal gætte én gang (første gang), sættes betingelsen i bunden af løkken.

Programmet kunne give følgend udskrift:

Gæt et tal mellem 1 og 100

Dit gæt: 70

Dit gæt 70 er for lavt

Dit gæt: 85

Dit gæt 85 er for højt

Dit gæt: 78

Dit gæt 78 er for lavt

Dit gæt: 80

Dit gæt 80 er for lavt

Dit gæt: 82

Dit gæt 82 er for højt

Dit gæt: 81

Tillykke, du har gættet tallet 81

## Opgave: Kommandohåndtering

Lav din Main-metode om sådan at brugeren skal indtaste en kommando. Den kommando brugeren indtaster skal bestemme hvilken af de metoder du har lavet der skal kaldes. Hvis brugeren skriver kommandoen ”stop”, skal programmet standse, ellers skal programmet udføre den metode der svarer til kommandoen og derefter bede brugeren om at indtaste en ny kommando.

På denne måde kan brugeren teste alle opgaverne af ved at skrive hvilken han ønsker udført og programmet standser først når han er færdig og skriver f.eks. "stop" eller "exit".

For at løse denne opgave skal du benytte Console.Write og Console.WriteLine til at fortælle brugeren hvilke muligheder han har, Console.ReadLine til at lade brugeren indtaste sin kommando. Gem kommandoen i en variabel. Det er dig der bestemmer hvilke kommandoer der skal være (en til hver metode i dit projekt).

Derefter kan du benytte if-sætninger til at undersøge hvad brugeren har skrevet og hvis det svarer til en af dine kommandoer, så udfør den metode der hører til kommandoen.

Lav en løkke rundt om koden sådan at den udføres så længe den kommando som brugeren indtaster, ikke er ”stop” (eller hvad du vil vælge som afslut-kommando).

Det vil også være en dog idé at kunne udskrive en oversigt over alle kommandoer. En metode der udskriver alle kommandoer kan så kaldes når man f.eks. skriver kommandoen "vis".

*Hint*: Hvis man benytter en do løkke, kan betingelsen være om kommandoen har været "stop". Da det ikke giver mening at tjekke om brugeren har skrevet kommandoen "stop" inden brugeren har indtastet nogen kommando, giver det god mening at have betingelsen i bunden - altså do-while.

## for løkker

Syntax:

for (<init>;<betingelse>;<count>) {<kode>}

Eksempel:

for (int i=0; i<110; i++) { … }

Når man på forhånd ved at man skal gennemløbe noget kode et bestemt antal gange, kan det være nemmere at bruge en for-løkke end en while-løkke. Det skyldes at man får hjælp af syntax-tjekkeren til at sikre at man husker at initialisere og tælle tællevariablen op – den der holder styr på hvor mange gange vi har udført koden inden i løkken. Det man skal have fokus på omkring for-løkker er at udfylde hovedet korrekt, altså det der står i parentes efter nøgleordet for.

Et simpelt lille eksempel gør det nemmere at forklare:

for (int i=0; i<10; i++)

{

Console.WriteLine(i);

}

Der er tre dele i hovedet af en for-instruktion, de tre dele er adskilt af semikolon.

* Den første del er initialisering, det udføres inden første gennemløb af løkken. Denne bruges ofte til at sætte tællevariablen til 0 (væn dig til at tælle fra 0, alting bliver nemmere på den måde).
* Den anden del er betingelsen for at fortsætte, det tjekkes ved starten og inden hver gennemkørsel af kode-delen. Når betingelsen er falsk fortsætter programmet efter løkken.
* Den sidste del udføres efter hver gennemkørsel af kode-delen. Denne bruges ofte til at sørge for at tælleren opdateres.

Det er muligt at løse samme opgaver med for som med while, men det normale er at for-løkker bruges når man skal lave en løkke der løber igennem en række tal fordi de er nemme at skrive op og hurtige at overskue når man læser koden, mens while-løkker af de fleste ses som mere overskuelige ved mere komplekse løkker fordi strukturen er nemmere at overskue. Tommelfingerregel: Hvis programmet på forhånd ved hvor mange gange den skal gentage en handling, så benyt for, ellers while.

Eksempel: Nedenstående er en løsning af gange med heltal ved hjælp af plus. Vi udnytter at 7\*8 er det samme som 7+7+7+7+7+7+7+7 (eller 8+8+8+8+8+8+8).

// Gange to positive heltal med hinanden

Console.Write("Indtast første tal der skal ganges: ");

int tal1 = Convert.ToInt32(Console.ReadLine());

Console.Write("Indtast andet tal der skal ganges: ");

int tal2 = Convert.ToInt32(Console.ReadLine());

int res = 0;

for (int i = 0; i < tal2; i++)

{

res += tal1; // samme som: res = res+tal1

}

Console.WriteLine($"{tal1} gange {tal2} giver {res}");

Programmet giver følgende brugerdialog.

Indtast første tal der skal ganges: 8

Indtast andet tal der skal ganges: 7

8 gange 7 giver 56

## Opgave: Jeg må ikke stjæle

Lav en metode hvor brugeren indtaster en sætning og hvor mange gentagelser han vil have. Programmet skal herefter skrive sætningen det antal gange som brugeren har ønsket.

Hvilken sætning: Jeg må ikke stjæle.

Hvor mange gentagelser: 3

Jeg må ikke stjæle.

Jeg må ikke stjæle.

Jeg må ikke stjæle.

## Opgave: Fødselsdag (lidt kringlet at få til at fungere helt rigtigt)

Lav en metode hvor brugeren kan indtaste sin alder og sit køn, og programmet så på skærmen udskriver

Hun blev 1 år, hurra, (Uden ’og’ - begynd med stort)

og hun blev 2 år, hurra,

og hun blev 3 år, hurra,

…

Osv. indtil den til sidst skriver:

og hun blev 21 år, HURRAAAA …

Hvis man f.eks. indtaster K som køn og 21 som alder.

**Bemærk**: Første og sidste linje afviger fra de øvrige (mange) linjer. Hvis alderen er 1 år, skal kun sidste linje vises, men da det også er den første linje, skal der ikke være noget ’og’.

## Opgave: Division

Hvis man kan gange ved hjælp af plus, hvordan kan man så implementere division?

Prøv at lave en metode som foretager heltalsdivision mellem to tal. Heltalsdivision vil sige at finde ud af hvor mange hele gange et tal går op i et andet. For eksempel er 7 div 2 = 3, 12 div 5 = 2, 10 div 5 = 2.

Overvej grundigt hvilken type løkke der giver mest mening at benytte.

Hint: Hvis du ikke kan gennemskue hvordan opgaven kan løses, kan du finde inspiration i eksemplet i afsnit 6.1.

## foreach løkker

Denne løkkestruktur benyttes til at gennemløbe arrays, lister eller tilsvarende datastrukturer. Vi har ikke præsenteret disse datastrukturer endnu, så denne løkkekonstruktion behandles i afsnit 8.5 nedenfor. Den er kun nævnt her for fuldstændighedens skyld, for at sikre at alle løkkestrukturer er nævnt i dette afsnit.

Syntaksen for foreach-løkker ser sådan ud:

foreach (<type> <name> in <list>) {<kode>}

Se afsnit 8.5 for en egentlig introduktion til foreach-løkker.

# Arrays

Indtil nu har vi arbejdet med enkeltstående variabler. Ofte har man dog en hel samling af værdier i spil som det ikke giver mening at gemme i deres egen variabel. Et eksempel kunne være deltagere i et spil. Vi ved ikke på forhånd hvor mange deltagere der er, så det vil være svært at oprette det rigtige antal variabler til at holde deltagernes navne, point etc. Ydermere er der sikkert mange operationer vi er interesseret i at foretage på alle spillere, og hvis de alle havde deres egen variabel til f.eks. navn skulle vi skrive den samme kode flere gange, én gang for hver forskellig variabel. Derfor ville det være godt hvis vi kunne gemme alle spillere i én variabel. Denne type af variabler der kan indeholde flere værdier kaldes containere og deres forskellige klasser kaldes for container classes.

## Array kort fortalt

Indbygget i sproget C# er typen array. Et array er en samling (tænk på det som en liste) af variabler hvor man i programmet angiver det maksimale antal elementer. Det betyder at man ved oprettelse skal kende det maksimale antal. Til gengæld bliver programmerne hurtige når man bruger arrays frem for de mere komplekse .Net containers (se nedenfor).

## Oprettelse af arrays

Syntaksen for arrays er arvet fra C, den er lidt speciel – nøgleordet *array* findes ikke i C#, man erklærer i stedet et array ved at benytte syntaksen [].

<type>[] <name>;

Man erklærer altså at en variabel skal være et array ved at skrive [] efter typen, så får man erklæret variablen som et array af denne type.

Eksempel:

string[] playerNames;

Nu har vi erklæret en variabel af typen array af string.

Et array adskiller sig fra andre indbyggede datatyper på den måde at vi ikke bare kan erklære et array og så begynde at bruge det. Vi skal have reserveret noget hukommelse til det i computeren. Det gør man sådan:

playerNames = new string[10];

Denne instruktion opretter et array med 10 elementer af typen string og sætter vores variabel til at *referere* det hukommelsesområde. Efterhånden som vi kommer dybere ned i C# skal vi se hvordan referencer bliver et vigtigt omdrejningspunkt i programmering.

Indtil videre er det ikke nødvendigt at forstå begrebet *reference*, bare husk at man laver et array ved at erklære det og oprette det. Som ved andre variabler kan det ske på én linje:

string[] playerNames = new string[15];

Denne linje erklærer og opretter et array af string med længde 15 og tilknytter det til variablen playerNames af typen *array af* string (hvilket skrives string[]).

## Brug af arrays

Man bruger arrays ved hjælp af firkantede ramme-tegn []. Eksempel:

playerNames[3] = "Claus";

Denne linje indsætter værdien "Claus" i det element i arrayet *der er indekseret af tallet 3*. Det er en noget kringlet formulering. Hvorfor ikke bare skrive ”det tredje element”? Fordi den formulering der ville være *forkert*. Den rigtige mundrette formulering ville være at skrive det fjerde element, men den formulering forvirrer indekset på det fjerde element er 3. Som nævnt tidligere, så gælder der i familien af C-sprog at *alting starter med* 0. Det vil sige at det første element ligger i indeks 0, det andet i indeks 1 osv. Det højeste indeks i playerNames som erklæret ovenfor er således 14. Det kan virke forvirrende i starten, men man vænner sig hurtigt til det.

Man kan benytte beregnede værdier til indeksering i arrays – ofte benytter man en indekseringsvariabel af typen int, som variablen i, i eksempelt nedenfor.

array[i] = Console.ReadLine();

Hvis man vil finde ud af hvilken værdi der ligger på et bestemt indeks i arrayet, kan man tilgå arrayet på tilsvarende måde:

Console.WriteLine(names[i]);

Nogle gange kan man have brug for at oprette et array med et fast sæt af værdier, f.eks. et array der indeholder alle lovlige karakterer i karakterskalaen. Så kan man erklære arrayet, oprette det og indsætte værdierne i ét hug:

int[] karakterer = {-3, 0, 2, 4, 7, 10, 12};

## Opgave: Lidt array-gymnastik

Lav en metode med et array af navne som er defineret direkte i koden. Derefter skal brugeren kunne indtaste et indeks og programmet skal udskrive navnet på den person der har det pågældende indeks.

Lav eventuelt en løkke, sådan at brugeren kan blive ved med at skrive indekser og få svar.

# Løkker og arrays

Løkker bruges meget ofte i sammenhæng med arrays. Man kan gennemløbe et array for at foretage samme handling på alle elementer, eller alle elementer der opfylder et eller andet krav. Med indeksering kan man nemt gennemløbe et array med en for-løkke:

for (int i = 0; i < array.Length; i++)

{

Console.WriteLine(array[i]);

}

Denne måde til at løbe et array igennem bruges rigtig ofte, men nedenfor skal vi se hvordan man med foreach endnu nemmere kan gennemløbe et array hvis man ikke har brug for at indekset til noget andet.

## Opgave: Spillernavne

Lav en metode hvor man kan indtaste navnene på deltagere til et spil 500. Man kan *maksimalt* indtaste 10 deltagernavne. Navnene skal gemmes i et array af string.

Det må gerne være sådan at brugeren selv kan bestemme hvor mange navne han vil indtaste.

Hint: Brug en for løkke og

## Hvordan man løser opgaver med løkker - brug en kandidat

Hvis man skal finde ud af noget om det data man har registreret i et array, kan man lave en løkke der gennemløber arrayet og undersøger det man nu vil undersøge.

Det kan godt virke lidt uoverskueligt når man pludselig løber ind i at skulle finde ad af f.eks. hvem der har fået flest point i et spil - hvordan gør man lige det?

Her kommer en kort introduktion til hvordan man kan finde bestemte oplysninger ved hjælp af en såkaldt *kandidat*. Det er nemmest at forstå metoden gennem et eksempel:

Eksempel: Find det største tal i et array af heltal

1. Opret en variabel der skal være kandidat til løsningen
2. Lav en løkke der gennemløber arrayet
3. For hver iteration, check om dette element i arrayet er en bedre kandidat en den nuværende kandidat. Hvis den er, sæt kandidaten til det nuværende element
4. Når løkken er færdig er kandidaten en bedre kandidat end nogen af de andre elementer, derfor er kandidaten svaret på opgaven

Kodeeksempel:

int[] numbers = { 14, 15, 12, 9, 18, 11, 15, 17, 6, 12, 15 };

int kandidat = 0;

for (int i = 0; i < numbers.Length; i++)

{

if (numbers[i] > kandidat)

{

kandidat = numbers[i];

}

}

Console.WriteLine($"Det største tal er {kandidat}.");

## Opgave: Bowling

Lav en metode hvor brugeren indtaster navn og antal point for et antal deltagere i en konkurrence (det kunne f.eks. være bowling).

Derefter skal metoden gennemløbe alle deltagere og finde ham/hende med flest point og færrest point, ham/hende med det længste navn og det korteste navn.

Bemærk: Programmet skal kunne svare på *hvem* der har vundet, ikke kun hvor mange point vedkommende har fået. Det betyder at man kan lade sig *inspirere* sig af kodeeksemplet ovenfor, men ikke bare nøjes med at kopiere det.

## Ekstraopgave: Bowling

Denne opgave kan eventuelt springes over - snak med din underviser.

Hvis der er flere der har samme pointtal eller navnelængde, skal alle disse deltagere udskrives.

## foreach-løkker

Ud over for-løkker, kan man også gennemløbe arrays og andre list-typer med foreach-løkker. foreach-instruktionen har følgende syntaks:

foreach (type name in list) {<kode>}

foreach-løkker kan bruges til nemt at gennemløbe arrays og andre list-typer. I eksemplet afsnit 8.1 ovenfor hvor arrayet gennemløbes og hvert element skrives ud er det eneste tællevariablen i bruges til at slå op i arrayet. I sådanne tilfælde er det nemmere at benytte foreach:

foreach (string elem in array)

{

Console.WriteLine(elem);

}

Denne måde at løbe lister igennem på gør koden meget nem at læse.

foreach vil gennemløbe alle araryets pladser, også dem der ikke er anvendt. Det kan gøre det lidt problematisk at benytte foreach. Senere når vi skal arbejde med List i stedet for array, bliver det mere naturligt at anvende foreach.

## Opgave: Udskriv alle navne med foreach

Lav en lille programstump der udskriver navnene (ikke deres point) på alle bowlingspillerne fra opgave 8.3.

Ville man kunne udskrive både navn og point med en foreach-løkke? Hvilken problematik løber man ind i hvis man vil have både navn og point udskrevet?

# Methods udvidet

Som vi så i afsnit *2 Methods grundlæggende* kan man kalde (aktivere) metoder i sin kode.

I dette afsnit skal vi se lidt mere på hvad man kan med methods og hvordan man kan bruge methods til at strukturere sit program.

I dokumentet [Erklæringer i C-Sharp](https://drive.google.com/open?id=1iXW8fN6KYML_8YVPosyAzmtforfq86W8) findes en præcis definition på hvordan methods erklæres.

## Parametre

Lad os først betragte et eksempel på en metode som vi har set før:

public static void Hello()

{

Console.WriteLine(”Hello”);

}

Denne metode udskriver ordet ”Hello” på console-skærmen. Men hvad hvis man nu gerne ville have metoden til at sige hej til nogen bestemt? Det ville være smart hvis man kunne sende et navn (f.eks. "Claus") med til metoden og der så blev skrevet "Hej Claus" på skærmen. Hvis man tænker at noget burde være muligt i et programmeringssprog, så er det ofte også tilfældet. Det gælder også her.

Som vist i dokumentet [Erklæringer i C-Sharp](https://drive.google.com/open?id=1iXW8fN6KYML_8YVPosyAzmtforfq86W8), så erklæres methods sådan:

<access> <type> <name>(<declaration parameters>) {<body>}

Det interessante i denne sammenhæng er det der står i parentesen efter method-navnet: declaration parameters. Her angiver man hvilke parametre man skal sende med når man vil kalde metoden. I vores eksempel er vi interesseret i at programmet skal sende et navn, dvs. en string, derfor erklæres metoden sådan:

static void Hello(string hvem) {…}

Metoden Hello(…) forventer nu at modtage en string-værdi når den bliver kaldt værdien bliver gemt i en lokal variabel hvem der kun eksisterer inde i metoden. Nu kan vi implementere metoden:

static void Hello(string hvem)

{

Console.WriteLine($"Hej {hvem}, dejligt vejr.");

}

Hvis nu metoden kaldes sådan her: Hello("Claus");

Hvis man så indtaster sit navn (her 'Claus'), får man følgende udskrift på skærmen::

Hej Claus, dejligt vejr.

Eksempel:

static void HelloMan()

{

Console.Write("Indtast dit navn: ");

string navn = Console.ReadLine();

Hello(navn);

}

Flere parametre angives med komma imellem. Eksempel:

static void Hello(string hvem, string hvad)

{

Console.WriteLine($"Hej {hvem}, dejlig(t) {hvad}.");

}

## Opgave: HelloMan udvidelse

Udvid HelloMan-metoden ovenfor sådan at metoden både beder om navn og noget brugeren gerne vil have ros for og som derefter roser brugeren for dette. Programmet kunne f.eks. have en skærmdialog som den nedenfor:

Indtast dit navn: Claus

Hvad vil du gerne have ros for (ét ord)? hår

Hej Claus, dejlig(t) hår.

## Opgave: method parametre (BMI)

Med udgangspunkt i *5.7 Opgave: BMI*:

Lav en ny *metode* der hedder UdskrivBMI (eller et andet sigende navn) og som tager et BMI-tal som *parameter*[[11]](#footnote-11) og udskriver om personen er overvægtig, normalvægtig eller undervægtig.

Tilret din BMI-løsning fra 5.7 sådan at den kalder din nye *metode*, eller, nok bedre lige nu, lav en ny *metode* der gør det samme (men uden at ændre i den gamle metode fra 5.7). På denne måde har du begge løsninger liggende så man kan sammenligne dem.

## Flere parametre (BMI)

Tilret metoden fra 9.3 sådan at *metoden* nu tager to *parametre*: Køn og BMI.

Tilret programmet sådan at det bruger den nye metode.

## Returværdi

I princippet *returnerer* alle *methods* en værdi til den der kalder *metoden*. En metode der er erklæret void returnerer dog ingen værdi. Det er den slags metoder vi har arbejdet med indtil nu.

En metode der returnerer en værdi, kaldes også en funktion.[[12]](#footnote-12)

Eksempel:

static **int Max(int a, int b)**

{

if (a > b)

{

return a;

}

else

{

return b;

}

}

Denne *metode* tager to heltal som *parametre* og *returnerer* et heltal (det største af de to parametre).

Instruktionerne return a; og return b; sørger for at metoden *returnerer* den rigtige værdi.

Så hvis *metoden* kaldes sådan: Max(3,8);, *returneres* værdien 8. Metoden Max kan man nogle gange gøre Det kan man bruge i sit program, i denne stump kode bruges Max til at sikre at vi ikke kommer til at dividere med 0 (nul):

Gennemsnit = sum / Max(antal, 1);

Sum og antal er variabler i programmet.

## Opgave: BMI funktion

Lav en ny method som tager to *parametre*, vægt og højde og ud fra de to værdier beregner og returnerer BMI.

Tilret din BMI-kode sådan at den benytter den nye funktion (aka den nye method).

## Opgave: Rest ved division

Lav en ny methode: int Rest(int dividend, int divisor) der *returnerer* rest ved heltalsdivision af de to tal den får ind som parameter.

Hvis du synes opgaven lyder bekendt, så er det rigtigt nok. I opgave 6.9 løste du nærmest samme problemstilling, så du kan kopiere den del kode derfra.

Lav herefter en *anden* metode til aftestning af Rest, hvor brugeren kan indtaste to tal og som så udskriver resten ved division (den rest der er tilbage når man dividerer det første tal med det andet).

1. Alle methods bortset fra void-methods og constructors, returnerer en værdi. Værdien skal ses som *resultatet* af at udføre metoden, metoden count() på bl.a. string, returnerer antallet elementer i strengen. [↑](#footnote-ref-1)
2. En begrænset mængde hukommelse sættes af til heltal. Derfor kan variabler af typen int indeholde værdier mellem -2147483648 og 2147483647, altså ikke alle heltal – for alle praktiske formål er dette dog ikke en begrænsning. [↑](#footnote-ref-2)
3. Strengen "-" er den tredje streng i udtrykket. [↑](#footnote-ref-3)
4. I programmeringssprog i C-familien bruger man ofte ’++’ til at tælle en variabel op med én, så ofte skriver man count++; fordi det er kortere end count = count + 1; [↑](#footnote-ref-4)
5. Teknisk set bliver der smidt (throw) en undtagelse (exception) som får programmet til at crashe med mindre undtagelsen bliver fanget et sted. Vi kommer til at arbejde med håndtering af exceptions senere. [↑](#footnote-ref-5)
6. Når man har flere udgaver af samme metode, dvs. med samme navn men forskellige parametertyper, kaldes det method overloading. Teknisk set er der tale om forskellige metoder, men overloading bruges ofte til at kunne foretage ”samme” handling ud fra forskellige situationer (afhængigt af hvilke informationer man har til rådighed). [↑](#footnote-ref-6)
7. Tallene er ikke korrekte, de passede i år 2018. [↑](#footnote-ref-7)
8. Sandhedsværdien af dette udtryk er usikker ;-) [↑](#footnote-ref-8)
9. Teknisk set er der her tale om to instruktioner, else og if(<condition>), men det er nemmest indtil videre at tænke på det som én instruktion. [↑](#footnote-ref-9)
10. [Læs evt. mere om tilfældige tal i C# her](../Random():%20http:/www.dotnetperls.com/random). [↑](#footnote-ref-10)
11. Bemærk at forskellen på denne opgave og opgave 5.7 er at vi nu deler opgaven op i mindre dele som hver løses i deres egen metode. Det betyder at vi får et program der er bedre struktureret og nemmere at læse og ændre i senere. [↑](#footnote-ref-11)
12. Nogle teoretikere anser kun en method som værende en funktion hvis den *ikke har sideeffekter*, dvs. hvis den ikke ændrer programmets opførsel gennem andet end den værdi der returneres. [↑](#footnote-ref-12)