

# Grundlæggende programmering

It & Data, Odense

kne - juni 2019

SYDDANSK  
ERHVERVSSKOLE



# Generel information

- Faget **grundlæggende programmering** har en varighed på 10 skoledage og afsluttes med en standpunktskarakter for faget.
- Faget er opdelt i to dele:
  - **Teoretisk del**, hvor læreren giver oplæg eller kommer med eksempler på projektor eller tavle.
  - **Praktisk del**, hvor eleven arbejder selvstændigt med de stillede opgaver.
- Faget afsluttes med en individuel evalueringssamtale med læreren, hvor eleven demonstrerer de færdige opgaver og læreren stiller uddybende spørgsmål, denne evaluering er tidssat til ca. 10 min.
- Der kan forventes hjemmearbejde i det omfang det er nødvendigt.



# Niveauer og evaluering

- Faget **grundlæggende programmering**, bedømmes på **3** niveauer.: **Rutineret**, **Avanceret** og **Ekspert**.
- Det er **valgfrit**, hvilket niveau man vil evalueres på, men alle starter som udgangspunkt på **Rutineret**.
- Des højere **niveau**, des højere **selvstændighedsgrad** forventes, samt **redegørelse** for valg og beslutninger skal kunne **uddybes** og **understøttes**.
- Inden evaluering, skal man tage stilling til hvilket niveau man vil bedømmes på.



# Introduktion til programmering

*Programmeringssprog, Syntax, Kodekonventioner,  
Udviklingsmijø, Compiled og Interpreted software*

# Kodekonventioner og programmeringssprog

- Der findes mange forskellige **programmeringssprog**, gamle som nye, men fælles for dem alle er, at de har hver sin måde de kodes (skrives på). Dette kaldes sprogets **syntax**. Mange sprogs **syntax** minder om hinanden, med større eller mindre afvigelser, der adskiller dem.
- **Syntax'en** udgøres af den **tegnSaetning();**, sproget bruger til at identificere hvad koden udfører af **kommandoer**. De fleste udviklingsværtøjer, har indbygget **fejlvise** af forkert **syntax** og foreslår evt. hvad fejlen er og måske rettelser her til.
- Alle programmeringssprog har en **kodekonvention**, der er beskrevet i de forskellige sprogs **dokumentation** eller manualer. En **kodekonvention** er et **regelsæt** for "**best practice**" i forhold til mappe/filstruktur, navngivning (naming convention), indentering, kommentarer, white space, praktisering og principper m.m.  
*Find det respektive sprogs officielle coding convention eller coding standard på nettet.*

# Programmering og udviklingsmiljø

- Når vi arbejder med programmering arbejder vi ofte i et **IDE** eller i en **Editor**. Begge er programmer man installerer på sin computer. Disse programmer er en del af ens **udviklingsmiljø**. Andre dele kan være: webbrowser, server, source-control, versionering m.fl.
  - **IDE** står for **I**ntegrated **D**evelopment **E**nvironment og er et avanceret program, der ofte er skræddersyet til et specifikt programmeringssprog eller flere. Kendetegnet for et **IDE**, er at det har indbygget **debug** funktionalitet, der anvendes når vi fejlsøger i vores kode. Ofte sørger vores IDE også for at kompilerer (compile) vores kode fra et højniveau programmeringssprog til et lavniveau sprog som maskinkode/binærkode **0000 0101 0011 1001**.
  - **Editor** er et "mindre" avanceret udviklings-progammel, forstået at her er der sjældent mulighed for debugging og der er som regel ikke indbygget en compiler til de sprog der skal kompileres. Det minder mere om et tekstdredigeringsværktøj som MS Word, bare med programmering for øje.
  - **Fælles** for begge udviklingsværktøjer er, at de ofte har IntelliSense også kaldet code-completion og auto-complete, hvilket betyder at programmet kommer med forslag til det kode vi er i gang med at skrive. Begge fremhæver også forskellige kommandoer i **farveKoder**, **typografier** og indentering, så vi nemmere kan danne os et overblik over vores kode.
- **Et godt udviklingsmiljø højner produktivitet og kvalitet.**

# Anbefaling!

I mit udviklingsmiljø, har jeg altid installeret et godt  
**IDE**  
og en god  
**EDITOR**

**HINT!** Vælg et IDE, der passer til det programmeringssprog, du skal udvikle i.

[https://en.wikipedia.org/wiki/Comparison\\_of\\_integrated\\_development\\_environments](https://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments)

# Opgave

- *Installer **Visual Studio** (IDE)*
- *Installer **Visual Studio Code** (Editor)*
- *Introduktion til VS*

# Interpreted vs Compiled software

- Nogle programmeringssprog bliver kompileret (**Compiled**) og andre er såkaldte oversatte (**Interpreted**) sprog.
  - **Kompileret software** er programmer, der er skrevet i et højniveau programmeringssprog som *C#*, *C++*, "**Java**" eller lignende. Før softwaren kan kører, bliver den **kompileret** af sprogets "motor" til et lavniveau sprog som maskinkode/binærkode **0000 0101 0011 1001**. Kompileren oversætter kildekoden til maskinkode, men først efter en gennemgribende bug-evaluering. I tilfælde af fejl i koden, kan programmet ikke kompileres og fejlmeddelelser gives. Kompileret software distribueres i kompileret udgave (executables) og kildekoden er dermed beskyttet.
  - **Interpreted software** er programmer, der distribueres i deres højniveau kildekode og først ved afvikling af programmet oversættes af sprogets motor (runtime engine) til maskinkode. Sprog som *PHP*, "**Java**" og mange andre, kan afvikles som Interpreted.

**Begge metoder har fordele og ulemper og i dag kan mange sprog både kompileres eller tolkes som Interpreted.**

# C#

(*udtales; 'C-Sharp'*)

# Introduktion til C#

- Udviklingen af programmeringssproget **C#** blev startet i 1999 af Microsoft under ledelse af danskeren **Anders Hejlsberg**, der stadig er ledende i udviklingen af sproget.
- C# er et **objektorienteret programmeringssprog (OOP)**, hvilket i vil blive introduceret for i faget Objektorienteret programmering. For nu, skal i blot vide at C# er et OOP.
- **C#** er baseret på en **C-syntax**, fra det aldrrende programmeringssprog C og er desuden et **Microsoft .NET** programmeringssprog.
- **.NET**(Udtales; Dot Net) **Framework'et**, er et såkaldt Framework Class Library (**FCL**) bestående af en masse pre-udviklet funktionalitet, der tilsammen med en motor kaldet Common Language Runtime (**CLR**) udgør frameworket. CLR håndterer kommunikation mellem applikation og operativsystem ved at oversætte koden. Frameworket håndterer også allokering/frigivelse af hukommelse (garbage collection), sikkerhed, debug og exception handling.

# C# Syntax og struktur

- C-syntax.
- C-syntax'en kendetegnes ved at alle kommandoer afsluttes med tegnet ; (semikolon).

Eks.

```
string navn = "Peter";
```

// Her tildeles en **streng** variable værdien **Peter**, men det sker først i det tråden læser tegnet ;

- **Spørgsmål: Hvad er en tråd?**
- **Indentering** i C# anvender almindelig **Child/Parent princip**, som vi kender det fra eks. HTML.  
Indentering betyder blot, at vi rykker kode ind under hinanden med linjeskift eller tabulator, for at vi nemmere kan læse og overskue koden.  
Indentering er vigtigt for mennesker, men har ingen praktisk betydning for en computer.
- Eftersom C# er objektorienteret, strukturerer den sin kode i klasser og metoder. Dette er blot til info.

# C# datatyper

- C# arbejder med fast angivelse af datatyper. Datatyper er den type af værdi man arbejder med: tekst, tal, tegn osv.
- Når vi arbejder med **tekst** angives det i **datatypen String**.
- Når vi arbejder med **hele tal** (uden decimaler) angives det i **datatypen Byte** eller **Integer**.
- **Reelle tal med decimaler**, kan angives i tre forskellige datatyper:  
**float (32bit)**, **double (64bit)** og **decimal (128bit)**.  
**Float** fylder mindst, men er også mest upræcis.  
**Double** fylder det dobbelte af float, men er mere præcis, dog ikke præcis nok til eksempelvis en bank.  
**Decimal** fylder mest, men er præcis nok til valutaangivelse, hvor sidste øre også tæller.

Se denne video, hvor valget af en forkert datatype fik fatale konsekvenser:  
<https://www.youtube.com/watch?v=EMVBLg2MrLs>

- Der findes mange andre datatyper og en komplet liste over datatyper i C# kan findes her: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/built-in-types-table>

# Opgave 1

- Lav et konsol program, der indsamler data om en bruger.
- Vi ønsker at kende flg. om en bruger
  - Fornavn
  - Efternavn
  - Alder
  - Køn (Skal angives med et m=mand, k=kvinde)
- Programmet skal spørge efter de relevante informationer og gemme dem i variabler med den korrekte datatype til den ønskede værdi.
- Køn skal vælges med datatypen char (karakter) k = kvinde, m = mand.
- Programmet skal spørge brugeren, hvor gammel brugeren mener man skal være, før man er gammel (eks. 60 år).
- Hvis brugeren er yngre end den angivne "gammel-alder", skal programmet fortælle, hvor mange år der er tilbage før han/hun bliver en gammel mand/kvinde og i hvilket år han/hun bliver gammel. Hvis brugeren er ældre eller har den samme alder som "gammel-alder", skal programmet fortælle brugeren, at han/hun ER en gammel mand/kvinde.

# Variable

- Variabler bruges til at gemme information, som vi kan refererer til og manipulerer med på mange måder.
- En variable er egentlig en allokering af hukommelse, hvori vi kan gemme en værdi. Værdien i en variable kan ændres og manipuleres med mange gange, mens et program afvikles (kører).
- En variable i C# skal altid deklareret med en passende datatype til den værdi variablen skal holde. I C# kan man dog konvertere datatypen på en variable under programafvikling, hvis værdien tillader det.
- Udover at vælge den korrekte datatype, skal vi også navngive vores variable med et passende og sigende navn.  
Eks.

```
int alder = 37;
```

(datatype er integer og navnet på variablen er "alder" -værdien er 37)

- Læg mærke til lighedstegnet (=) ovenfor, det er en såkaldt **operator**, der i dette tilfælde tildeler en værdi til variablen alder.

# Konstanter

- En konstant minder om en variable, men til forskel fra variabler kan konstanter ikke ændre deres værdi under programafvikling.
- Konstanter får deres datatype og værdi ved kompilering af et program og herefter er de helt låst og kan aldrig ændres uden at re-kompilerer programmet og omskrive kildekoden.
- Eks.

```
const double pi = 3.141592;
```

# Metoder (funktioner)

- I programmering, kan vi arbejde med metoder og når vi i C# ser et navn efterfulgt af parenteser (eks. `ToString()`), så ser vi faktisk et metodekald – altså en funktion, der eksekveres.
- Metoder er en måde hvormed vi kan opdele og afgrænse funktionalitet i mindre programstykker, der kan kaldes og genbruges efter behov. Dette er en del af begrebet om encapsulation (indkapsling), der handler om at afgrænse tilgængeligheden af vores kode. Det er også en måde hvormed vi kan strukturer vores kode, så den er mere letlæselig.
- I C# kan metoder være indbygget i .NET FrameWorket, som `ToString()`, men vi kan også lave vores egne metoder og eksekvere dem med et metodekald i vores program.  
Når vi laver vores egne metoder, giver vi dem et sigende navn, ligesom med variabler og konstanter.
- Metoder kan modtage parametre, der overføres ved metodekaldet.  
*Eks. void angivNavn(string navn){ ... } og angivNavn("Rasmus");*  
De overførte parametre kan anvendes inde i metoden, således metoden får data udefra, som den igen kan anvende i sin funktionalitet.
- Metoder kan have en returværdi i en angivet datatype og i C# angives denne i metodens angivelse. Hvis metoden ikke returnerer noget data, bruges ordet **void** i stedet for en egentlig datatype.
- ***TODO eks!***
- ***En metode kan kalde sig selv, hvilket kaldes rekursion.***
- ***Method overloading***

# Operatorer (1/4)

Med **operatorer** kan vi **tildele**, **sammenligne**, **beregne** og bruge **boolsk algebra** på værdier.

På de næste slides, gennemgår vi de mest anvendte operatorer.

For en komplet liste af operatorer, følg linket:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/operators>

## Matematiske operatorer:

Når vi anvender matematik i vores programmering, så skal vi huske de almindelige regneregler, som vi kender fra matematik timerne.

Eks. Gange og Divider kommer før Plus og Minus og man kan bruge **parenteser** for at **prioriterer** hvad der først skal udregnes.

Tegn	Udregning	Forklaring
+	$X = A+B;$ Addition	Vi ligger et tal til et andet tal
-	$X = A-B;$ Subtraktion	Vi fratrækker et tal fra et andet tal
*	$X = A*B;$ Multiplikation	Vi ganger et tal med et andet tal
/	$X = A/B;$ Division	Vi dividerer et tal med et andet tal
%	$X = A\%B;$ Modulus	Rest værdi efter division
++	$A++;$ Inkrementer	Læg 1 til værdien af A
--	$A--;$ Dekrementer	Træk 1 fra værdien af A

# Operatorer (2/4)

Relation	Eksempel A holder 10 B holder 12	Forklaring
<code>==</code>	A == B, er falsk	Hvis A har den samme værdi som B, er dette sandt
<code>!=</code>	A != B, er sandt	Hvis A IKKE har den samme værdi som B, er dette sandt
<code>&gt;</code>	A > B, er falsk	Hvis A holder en større værdi end B, er dette sandt
<code>&lt;</code>	A < B, er sandt	Hvis A holder en mindre værdi end B, er dette sandt
<code>&gt;=</code>	A >= B, er falsk	Hvis A holder en værdi, der er større eller det samme som B, er dette sandt
<code>&lt;=</code>	A <= B, er sandt	Hvis A holder en værdi, der er mindre eller det samme som B, er dette sandt

## Relationsoperatorer (Sammenligningsoperator)

Med Relationsoperatorer kan vi sammenligne værdier med hinanden på forskellige måder.

Ofte er det værdien, som en variable holder, der sammenlignes med en værdi eller værdien af en anden variable.

# Operatorer (3/4)

## Tildelings operatorer:

Med tildelingsoperatorer (assignment), kan vi manipulerer eller tildele en værdi til en variable.

Tegn	Eksempel <b>X holder 2</b> <b>Y holder 7</b>	Forklaring
= Assignment	X = Y; X = 7	Vi tildeler værdien Y til variablen X
+= Addition	<b>X += Y;</b> (X = X + Y) (X = 9)	Vi ligger Y til variablen X eksisterende værdi og gemmer i X
-= Subtraktion	<b>X -= Y;</b> (X = X - Y) (X = -5)	Vi trækker Y fra variablen X eksisterende værdi og gemmer i X
*= Multiplikation	<b>X *= Y;</b> (X = X * Y) (X = 14)	Vi ganger værdien af X med Y og gemmer i X
/= Division	<b>X /= Y;</b> (X = X / Y) (X = 0,285..)	Vi dividerer X med Y og gemmer i X
%= Modulus	<b>X %= Y;</b> (X = X % Y)	Rest værdi efter division med X og Y gemmes i X
Der findes flere, se dem her: <a href="https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/operators">https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/operators</a>		

# Operatorer (4/4)

Relation	Eksempel X holder 3 Y holder 7	Forklaring
& Logisk AND	$1 \& 1 = \text{TRUE}$ , $1 \& 0 = \text{false}$ $(Z = X \& Y)$ $(Z = 3)$	Hvis to værdier er ens er et udsagn sandt ellers falsk.
 Logisk OR	$1   1 = \text{true}$ , $1   0 = \text{true}$ $(Z = X   Y)$ $(Z = 7)$	Bare én værdi er sand, er udsagnet sandt også hvis begge er sande.
^ Logisk XOR	$1 ^ 1 = \text{false}$ , $1 ^ 0 = \text{true}$ $(Z = X ^ Y)$ $(Z = 4)$	Hvis KUN én værdi er sand, er udsagnet sandt (Exclusive)

## Logiske operatorer (Boolsk algebra)

Med logiske operatorer kan vi sammenligne værdier med hinanden på forskellige måder.

Ofte er det værdien, som en variable holder, der sammenlignes med en værdi eller værdien af en anden variable.

# Betingelser (Kontrolstruktur)

- I al programmering arbejder vi ofte med betingelser (statements). Betingelser handler om at udvælge hvad der skal ske, hvis et givent scenarie er sandt eller falsk.
  - Betingelser bruges i flere koder som: *if*, *while*, *for* osv., men i langt de fleste tilfælde, henvises til **if** sætningen, der i sin grundform altid søger om et givent udsagn er sandt.
  - If kan kombinere kan kombineres med else if og med else, som vist på eksemplet.
- ```
int age = 37;

if(age <= 20){
    Console.WriteLine("Du er ikke helt myndig!");
}
else if (age >= 21 && age < 65){
    Console.WriteLine("Du er myndig");
} else {
    Console.WriteLine("Du er pensionist");
}
```

# Switch

- En switch kan minde lidt om en "if else" betingelse, men den store forskel ligger i, at switchen ikke validerer for andet end én værdi. I switchen kan vi altså ikke opsætte komplekse betingelser.
- Vi vælger altid en switch fremfor "if else" syntaxen, da en switch i langt de fleste tilfælde er hurtigere at afvikle.
- En anden grund, er at switch'en ser mere overskuelig ud, end en lang række af "if else" betingelser og det er nemmere at overskue og læse.

```
string color = "sort";

switch(color)
{
    case "hvid":
        Console.WriteLine("Farven er hvid");
        break;
    case "sort":
        Console.WriteLine("Farven er sort");
        break;
    default:
        Console.WriteLine("En anden farve er valgt");
        break;
}
```

# Iteration (gentagelse)

- Hvis vi ønsker **iterere** en handling **X antal gange** eller så længe en betingelse er opfyldt, så er det **loops** (løkker) vi anvender.
- Brug **loops** til at tælle, lave rækker og kolonner, gøre noget for hvert item, så længe en betingelse er opfyldt eller til at udtrække hver datarække i et resultat fra en dataforespørgsel.
- I C# kan vi vælge en af **4 loops** til vores **iterationer**.
  - **for**
  - **While**
  - **do while**
  - **Foreach (til lister)**
- Hvert af ovenstående **loop** anvendes i specifikke scenarier, hvor de er mest praktisk.

# For loop

- Et **for loop** anvendes bedst, når vi ønsker at iterere noget, et **fast** og **forudbestemt antal gange**. Eks. Tælle til 10 eller 1.000.000.000
- Med **for loopet**, kan vi definere en tælle variable i selve deklarationen af løkken.

Eks. 1

```
for (int i = 0; i < 10; i++){  
    Console.WriteLine("i er lig med: " + i.ToString());  
}
```

Eks. 2

```
int i = 0;  
  
for (i = 0; i < 10; i++){  
    Console.WriteLine("i er lig med: " + i.ToString());  
}  
  
Console.WriteLine("efter løkken har kørt er i: " + i.ToString());
```

# Mini Opgave 2

Lav et program, der udskriver 1 til 10 tabellen, som vist nedenfor. (10 rækker og 10 kolonner)

Du må **KUN** anvende **for loops, variable, udregning, PadLeft** metoden og **output** med Console.WriteLine()

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10  |
| 2  | 4  | 6  | 8  | 10 | 12 | 14 | 16 | 18 | 20  |
| 3  | 6  | 9  | 12 | 15 | 18 | 21 | 24 | 27 | 30  |
| 4  | 8  | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40  |
| 5  | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50  |
| 6  | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60  |
| 7  | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70  |
| 8  | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80  |
| 9  | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90  |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

# While loop

- **While loopet** bruges bedst, til at **iterere** et **ikke kendt antal gange**. Eks. Så længe en betingelse er opfyldt.
- **PAS PÅ!** Med **infinit loops!** (uendelige løkker)

```
Random dice = new Random();
int res = dice.Next(1, 7);
int i = 1;

while(res != 6){
    res = dice.Next(1, 7);
    i++;
}
Console.WriteLine("Det tog #{0:D} kast, at slå 1 sekser", i);
```

# Do while loop

- **DO while loopet** er en afart af while loopet og bruges ligeledes bedst, til at **iterere et ikke kendt antal gange**.  
Forskellen er dog, at med do while, kører løkkens funktion, altid mindst én gang før betingelsen kontrolleres.

```
Random dice = new Random();
int res;
int i = 0;

do{
    res = dice.Next(1, 7);
    i++;

} while (res != 6);
Console.WriteLine("Det tog #{0:D} kast, at slå 1 sekser", i);
```

# Opgave 3

Lav et konsol-program, hvor det handler om at slå 6'ere med et valgfrit antal terninger.

Når programmet starter skal man indtaste et antal terninger.

Programmet skal så ”kaste” terningerne og tælle hvor mange kast, der skal til at slå ”rene” seksere (alle terninger viser 6 i samme kast).



# Array

- Et **array**, er i sin enkelhed en "variable", der kan holde mere end én værdi. -Det er en liste af værdier (Det er faktisk, et **objekt** af klassen **Array**).
- I C# skal vi angive længden af vores array instans i deklarationen. Længden er fast og kan ikke ændres i instansens levetid.
- Et array i C# er zero-indexed, hvilket betyder at et array med n værdier går fra 0-n.
- Et array i C# kan være **single-dimensional**, **multi-dimensional** eller **jagged**.
- Læs alt om arrays her: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/>

# Single-dimensional array

- Et single-dimensional array, betyder at vi har atøre med en enkelt liste af værdier.
- Hver værdi er indekseret med en key. Key'en er kronologisk ordnet fra 0 og inkrementerer med 1.
- Eks.** Vi ønsker at gemme tallene 2, 3, 8, 4 i et array eller en liste over arbejdsdage:

```
// Single-dimension  
int[] arr1 = new int[4];  
arr1[0] = 2;  
arr1[1] = 3;  
arr1[2] = 8;  
arr1[3] = 4;  
  
int[] arr2 = new int[] { 2, 3, 8, 4 };  
  
int[] arr3 = { 2, 3, 8, 4 };  
  
Console.WriteLine(arr3[2].ToString());  
  
string[] arbejdsDage = { "Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag" };
```

| Key | Value |
|-----|-------|
| 0   | 2     |
| 1   | 3     |
| 2   | 8     |
| 3   | 4     |

| Key | Value   |
|-----|---------|
| 0   | Mandag  |
| 1   | Tirsdag |
| 2   | Onsdag  |
| 3   | Torsdag |
| 4   | Fredag  |

# Multi-dimensional array

- Et array kan have mere end én dimension, hvilket betyder at hver værdi i et array, holder en liste af værdier i anden, tredje, fjerde dimension osv.
- Eks1. 2 dimensioner:** Vi ønsker at gemme 1 og 2 tabellen i et array med 2 pladser. Hver plads skal holde alle 10 værdier i de respektive tabeller:

```
// Multi-dimension, 2 dimesnioner
int[,] arr2d = new int[2,10] { {1,2,3,4,5,6,7,8,9,10},
                             {2,4,6,8,10,12,14,16,18,20} };
Console.WriteLine(arr2d[1,9]); // = 20
```

- Eks2. 3 dimensioner:**

```
// Multi-dimension, 3 dimesnioner
int[, , ] arr3d = new int[2,2,3]
{
    {
        { 1, 2, 3 },
        { 4, 5, 6 }
    },
    {
        { 7, 8, 9 },
        { 10, 11, 12 }
    }
};
Console.WriteLine(arr3d[1,1,2]); // = 12
```

| key | Value |       |
|-----|-------|-------|
| 0   | key   | value |
| 0   | 1     |       |
| 1   | 2     |       |
| 2   | 3     |       |
| 3   | 4     |       |
| 4   | 5     |       |
| 5   | 6     |       |
| 6   | 7     |       |
| 7   | 8     |       |
| 8   | 9     |       |
| 9   | 10    |       |

  

| 1 | key | value |
|---|-----|-------|
| 0 | 2   |       |
| 1 | 4   |       |
| 2 | 6   |       |
| 3 | 8   |       |
| 4 | 10  |       |
| 5 | 12  |       |
| 6 | 14  |       |
| 7 | 16  |       |
| 8 | 18  |       |
| 9 | 20  |       |

# Jagged array

- I et såkaldt jagged array, har vi mulighed for at gemme arrays i arrays. Det betyder at hvert element i et jagged array kan være af forskellig længde og dimension.
- Læs mere om det her: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/jagged-arrays>

# Foreach loop

- I et foreach loop, kan vi gennemløbe alle elementer i en liste.  
Listen skal bestå af elementer, der har arvet **Ienumerable**, som  
for eksempel et **array objekt**.
- **Eks.** Vi ønsker at udskrive alle ugens arbejdsdage, som er gemt i  
et array: *Mandag, Tirsdag, Onsdag, Torsdag, Fredag*

```
string[] arbejdsDage = { "Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag" };

foreach(string day in arbejdsDage){
    Console.WriteLine(day);
}
```

Mandag  
Tirsdag  
Onsdag  
Torsdag  
Fredag

# Opgave 4

- Lav en lommeregner konsol-applikation, hvor brugeren kan indlæse 2 tal samt en regneoperator fra tastaturet.  
Hele programmet kan i første omgang placeres i metoden Main().  
Output fra programmet skal være resultatet af den pågældende regneoperation.
- Prøv at udvide programmet, så brugeren kan regne videre på det sidste resultat, og selv vælge hvornår programmet afsluttes.
- Prøv også at flytte regneoperationerne til metoder, så metoderne kaldes med de 2 tal, der skal regnes på. Den aktuelle metode vælges ud fra den indtastede regneoperator.
- Udvid programmet med funktionalitet til en avanceret lommeregner.

|    |     |   |   |
|----|-----|---|---|
| 0  |     |   |   |
| AC | +/- | % | ÷ |
| 7  | 8   | 9 | × |
| 4  | 5   | 6 | - |
| 1  | 2   | 3 | + |
| 0  | ,   |   | = |

|                 |                |                |                |                |                   |    |     |   |   |
|-----------------|----------------|----------------|----------------|----------------|-------------------|----|-----|---|---|
| 0               |                |                |                |                |                   |    |     |   |   |
| (               | )              | mc             | m+             | m-             | mr                | AC | +/- | % | ÷ |
| 2 <sup>nd</sup> | x <sup>2</sup> | x <sup>3</sup> | x <sup>y</sup> | e <sup>x</sup> | 10 <sup>x</sup>   | 7  | 8   | 9 | × |
| $\frac{1}{x}$   | $\sqrt[2]{x}$  | $\sqrt[3]{x}$  | $\sqrt[y]{x}$  | ln             | log <sub>10</sub> | 4  | 5   | 6 | - |
| x!              | sin            | cos            | tan            | e              | EE                | 1  | 2   | 3 | + |
| Rad             | sinh           | cosh           | tanh           | π              | Rand              | 0  | ,   |   | = |

# Opgave 5

- En lille frø ønsker at krydse en vej. Frøen er positioneret på lokation: **X** og ønsker at komme til eller forbi lokation: **Y**. Frøen hopper altid den samme distance **D** i hvert hop.
- Lav et program, der kan tælle det minimum antal hop frøen skal udføre, for at nå dens mål.
- I programmet, skal du lave en metode, der modtager **3 parametre X, Y og D** og ud fra værdierne af disse **returnerer et heltal**, der viser **minimum** antal hop fra lokation **X** til en position, der er lig med eller større end lokation **Y**.
- **Eks.** Metoden modtager nedenstående værdier:  
 $X = 10$   
 $Y = 85$   
 $D = 30$   
Metoden returnerer her 3, fordi frøen positionerer sig som følger:
  - Efter første hop til position:  $10 + 30 = 40$
  - Efter andet hop til position:  $10 + 30 + 30 = 70$
  - Efter tredje hop til position:  $10 + 30 + 30 + 30 = 100$
- Du kan **antage** at:
  - **X, Y og D** er af typen **integer** og holder en værdi fra **1 til 1.000.000.000**
  - **X <= Y**
- **HUSK at lave en gennemgribende test af dit program! Det er sandsynligt, at der sendes direkte forkerte eller MEGET store tal til dit program.**  
**Lav gerne metoder til at teste dit program (UnitTest).**

# Opgave 6

- Du modtager et zero-indexed array **A**, der holder **N** antal værdier. Værdierne i arrayet, skal "rotere" mod højre **K** antal gange. Når et element flyttes ud af sidste index i arrayet, skal det flyttes til første index i arrayet, dermed én rotation.
- **Eks.**  
Rotationen med 1 index i array A = [3,8,9,7,6] er lig med [6,3,8,9,7]
  - Idéen er altså at rottere værdierne i array A mod højre K antal gange.
- Lav en metode, der modtager 2 parametre array **A** og int **K**. Metoden skal returnerer array **A**, hvor værdierne er roteret **K** antal gange mod højre.
- Du kan antage at:
  - **N** og **K** er af typen integer med værdier fra 0 til 100
  - Hvert element i array **A** er en integer med blandet værdier fra -1.000 til 1.000
- **HUSK at lave en gennemgribende test af dit program! Det er sandsynligt, at der sendes direkte forkerte eller MEGET store tal til dit program. Lav gerne metoder til at teste dit program (UnitTest).**

# Opgave 7

- Du modtager ét zero-indexed array **A** med **N** antal heltal værdier. **A** indeholder et **ulige** antal elementer. Hvert element kan parres med ét andet element af samme værdi, undtaget ét element.
- **Eks.**  
A[0]=9, A[1]=3 , A[2]=9  
A[3]=3 , A[4]=9 , A[5]=7  
A[6]=9
  - Index 0 kan parres med index 2 med værdien 9
  - Index 1 kan parres med index 3 med værdien 3
  - Index 4 kan parres med index 6 med værdien 9
  - Elementet med index 5 har værdi 7 og kan ikke parres
- Lav en metode, der modtager array **A**. Metoden skal returnerer værdien af det element, der ikke kan parres.
- I eksemplet ovenfor, vil metoden returnerer tallet 7.
- Du kan **antage** at:
  - **N** er en ulige integer med værdien 1 til 1.000.000
  - Hvert element i array **A** er en integer med blandet værdier fra 1 til 1.000.000.000
  - Alle værdier i **A** fromkommer et lige antal gange, med undtagelse af et eneste element
- **HUSK at lave en gennemgribende test af dit program! Det er sandsynligt, at der sendes direkte forkerte eller MEGET store tal til dit program.**  
**Lav gerne metoder til at teste dit program (UnitTest).**

# Opgave 8

- Du skal lave en konsol applikation. Applikationen skal være et regnespil, forstået på den måde, at programmet skal stille brugeren et regnestykke i flg. Discipliner: addition, subtraktion, multiplikation og division. Brugeren skal kunne vælge disciplin ud fra en konsol menu.
- Brugeren skal vælge et niveau for sværhedsgrad af regnestykkerne, hvorefter programmet stiller 10 regnestykker, som brugeren skal svare rigtigt på. Hvis brugeren ikke svarer rigtigt på regnestykket indenfor 3 forsøg, så skal programmet give svaret og gå videre.
- Hver gang brugeren stilles et spørgsmål, skal brugeren have mulighed for at vende tilbage til en hovedmenu, hvor disciplin vælges.
- Programmet kan kun afsluttes fra hovedmenuen.
- Programmet skal give point for rigtige svar og alt efter sværhedsgrad. Sværhedsgraden ligger i regnestykkets størrelser.
- **Eks.**
  1. tal fra 1 – 10
  2. tal fra 1 – 100
  3. tal fra 1 – 1.000
  4. tal fra 1 – 9.999
- Lav evt. en option med minus tal
- Lav evt. en option, hvor man kan træne eksempelvis en tabel, som 3 eller 5 tabellen.
- Man kan lave en highscoreliste, der gemmes/læses i en fil (evt. Csv fil).  
[https://www.tutorialspoint.com/csharp/csharp\\_text\\_files.htm](https://www.tutorialspoint.com/csharp/csharp_text_files.htm)
- **HUSK at lave en gennemgribende test af dit program! Det er sandsynligt, at der sendes direkte forkerte eller MEGET store tal til dit program.**  
**Lav gerne metoder til at teste dit program (UnitTest).**

# UnitTest