# C# Fundamentals for Absolute beginners

Mine egne kommentarer til kurset.

Link: <https://mva.microsoft.com/en-US/training-courses/c-fundamentals-for-absolute-beginners-16169?l=p90QdGQIC_7106218949>

Host: Bob Tabor

### 03. Creating your first C# program

* HelloWorld program
* End all commands using semicolon
* Mark strings “
* C# is case sensitive

### 04. Understanding you first C# program

* How to write code
  + Learn syntax
  + Learn what is already made, prebuild available functionality; .NET framework.
* Class Library
* Common Language Runtime [CLR]
  + Developer may focus on what the application should do, The CLR takes care of the what’s going on behind the scene.
  + Hardware, memory, …
  + Protection for the user. The app is running inside a “bubble”
* Linefeed, whitespace don’t bother in C#. ONLY “;” will terminate a command

### 05. Working with Code Files, Projects and Solutions

* Code files are organized into projects
* Project are compiled into single assembly
* A solution may contain of several assemblies

### 06. Understanding Datatypes and Variables

* Use camelCasing for increased readability
* Refactoring by [ctrl] + [.]
* C# is CaseSensitive!
* Intellisense may automatically pop up
  + Manual: [ctrl] + [space]
  + may autocomplete using marked alternative when typing [=] or [.] or [enter]
* Declare variables as you need then
  + As opposed to declare in separate line, and opposed to declaring in the top of code
  + Opt. give them initial values

### 07. the IF decision statement

* [==] means to evaluate if something is the same
* If, if else, else
* A variable is only available inside current scope {}
* One-line-code-block does not need curly braces.
* Var += something; is short for: var = var + something;
* If(var == value) ? result-if-yes; result-if-false;
* Writeline(“You won a {0}.”, var); //replace the {0} with the value of var.
* Console.WriteLine("You entered: {0}, therefor you won a {1}.", userValue, message); //works with several replacements like this

### 08. Operators, Expressions and Statements

* Basic building blocks:
  + Statements. Complete thoughts, like a sentence.
  + Statements are made of expressions
  + Expressions are made of operators and operands
  + Operands are the nouns. The things, the objects [classnames, ..]
    - Developer give these names
  + Operators are the verbs. Performs action [=, +=]
    - Developer need to learn these
* See project: “OperatorsExpressionsStatements” for code examples/ documentations

### 09. Iteration Statement

* i++: ++ is the iteration statement
* Breakpoint:
  + hover to monitor variables, change values of variables, change what code is executed next
  + [F10] step over the breakpoint and continue executing the code
  + [Continue] the program will execute up to the next breakpoint
  + Breakpoint settings -> Conditional breakpoint will “listen” for a condition come true for breaking.
  + Enable/Disable
* Codesnippets: start writing fex “for” and intellisense will show; press [tab]+[tab] to tell intellisense to generate a for loop template.

### 10. Understanding Arrays

* Syntax: int[] numbers = new int[5]; //create an array of type int. make space for 5 elements.
* Create arrays instead of separate variables
* Makes it easier to iterate
* New int[5] : explicitly make room for 5 elements
* New int[] { 2, 4, 6} : Let the compiler decide the length of the array
* Iteration by
  + for (int i == 0 ; i < 5 ; i++) : Access value using “array[i]”
  + foreach (string name in names) : Access value using temporary var “name”.
* “Array.Reverse(anArray)” :
  + Is a .Net framework method for the class “Array”. Takes an array as argument.
  + Part of System Nnamespace.
* More elegant way of working with “Arrays” of data is by using “Collections”, covered later in the course.

### 11. Defining and calling methods

* Related methods goes inside the context of a class
* Create a method instead of copy/past code
  + Easier to maintain
  + Less code
  + More readable
  + Give the method a describing name
  + The application becomes more readable
* Hardcode first, make dynamic afterwards
* A method should have a describing name
  + If a method gets several responsibilities -> split into separate methods
* String class:
  + .Concat : concatenate elements into one string
  + String.Format(“bla this {0} and bla that{1} ”, replacementA, replacementB) : usefull way of formatting or creating a string with dynamic content
* Overloaded method MUST have individual “signature”
  + Different number of arguments OR different types of arguments
  + Will show up in intellisence