C#

- Begrepp
  - Konkatenering
    - Console.WriteLine()
    - Console.ReadLine()
  - Konvertering (s 29-31)
    - Convert.ToInt32(string) eller int.Parse(string) — https://www.tutorialsteacher.com/articles/convert-string-to-int
    - Casting — Tvinga anta viss datatyp
    - Widening and Narrowing
  - Scope
    - Livslängd
    - The C# compiler also looks after the part of a program within which a variable has an existence.
    - The scope of a local variable is the block within which the variable is declared. You can declare a variable at any point in the block, but you must declare it before you use it.
  - Planeringsverktyg
    - Pseudokod
    - Aktivietsdiagram
  - Metadata — Information about information.
  - Prototyping
  - Object oriented
    - Encapsulation/information hiding
      - Define fields as private
      - Provide getter/setter methods as public
    - Inheritance
    - Polymorphism
  - Compiled or interpreted
  - Source file
  - Statement
    - Instruction to do something
    - Assignment statement
  - Libraries
    - A series of methods.
    - C# library.
    - Microsoft .Net Framework
  - Keywords
    - Keywords are reserved words predefined to the C# compiler. These keywords cannot be used as identifiers. However, if you want to use these keywords as identifiers, you may prefix the keyword with the @ character.
      - Reserved Keywords — Ex. do, const, case, new, ref, void
      - Contextual Keywords — Ex. add, alias, join, set
    - static — Makes sure that the method which follows is always present. The variable is part of the class and is always present.
    - void — Means that the method not return anything.
    - Main — Method where yor program starts running.
    - const — The value cannot be changed.
  - Testing
    - A very important part of program development.
    - Test out all possible states within the prgoram, including the all-important error conditions.
  - Identifiers
    - A name the programmer choose for something in the program.
    - All identifiers names must start with a letter.
    - After the letter you can have either letters or numbers or the underscore _ character.
    - Always give your variables meaningful names.
    - Use camel case. — firstName
  - Instructions and messages
  - Namespace
    - A namespace is a place where particular names have meaning.
    - Namespaces are all to to do with finding resources.
    - You can create your own namespaces.
    - Fully qualified name
      - System.Console.WriteLine("Hello World");
      - Once we have specified a namespace in a program file we no longer need to use the fully qualified name for resources from that namespace.
  - Classes
    - A class is a container which holds data and program code to do a paticular job.
    - In C# everything exist inside a class. A class is a container for a whole bunch of things, including methods.
    - Classes have members
      - A member may contain
        - Fields
          - Fields are variables declared at class scope. A field may be a built-in numeric type or an instance of another class. For example, a calendar class may have a field that contains the current date.

```
public class Person
{
   private DateTime _birthdate;

   public DateTime SetBirthdate(DateTime birthdate)
   {
      _birthdate = birthdate;
   }

   public DateTime GetBirthdate()
   {
      return _birthdate;
   }
}
```

        - Constants — Constants are fields whose value is set at compile time and cannot be changed.
        - Properties
          - Properties are methods on a class that are accessed as if they were fields on that class. A property can provide protection for a class field to keep it from being changed without the knowledge of the object.
          - A class member that encapsulates a getter/setter for accessing a field.
            - To create a getter/setter with less code.
            - Example

```
public class Person
{
   private DateTime _birthdate;
   public DateTime Birthdate
   {
      get { return _birthdate; }
      set { _birthdate = values; }
   }
}
```

              - Auto-implemented Properties

```
public class Person
{
   public DateTime Birthdate { get; set; }
}
```

        - Methods — Methods define the actions that a class can perform. Methods can take parameters that provide input data, and can return output data through parameters. Methods can also return a value directly, without using a parameter.
        - Events — Events provide notifications about occurrences, such as button clicks or the successful completion of a method, to other objects. Events are defined and triggered by using delegates.
        - Operators — Overloaded operators are considered type members. When you overload an operator, you define it as a public static method in a type. For more information, see Operator overloading.
        - Indexer
          - Indexers enable an object to be indexed in a manner similar to arrays.
          - A way to access elements in a class that represents a list of values.
        - Constructors — Constructors are methods that are called when the object is first created. They are often used to initialize the data of an object.
        - Finalizers — Finalizers are used very rarely in C#. They are methods that are called by the runtime execution engine when the object is about to be removed from memory. They are generally used to make sure that any resources which must be released are handled appropriately.
        - Nested types — Nested types are types declared within another type. Nested types are often used to describe objects that are used only by the types that contain them.
      - All types and type members have an accessibility level. The accessibility level controls whether they can be used from other code in your assembly or other assemblies. Use the following access modifiers to specify the accessibility of a type or member when you declare it.
        - **Access modifiers - a way to control access to a class and/or its members**
          - Why? - to create safety in our programs.
          - Why access modifiers are important
          - Black box metaphor
          - C# access modifiers
        - public
          - The type or member can be accessed by any other code in the same assembly or another assembly that references it.
          - Accessible from everywhere.

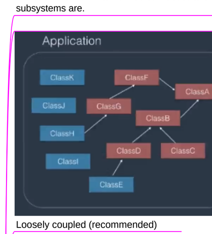
```
public class Customer
{
   public void Promote()
   {
   }
}
...
var customer = new Customer();
customer.Promote();
```

        - private
          - The type or member can be accessed only by code in the same class or struct.
          - Accessible only from the class.

```
public class Customer
{
   private int CalculateRating()
   {
   }
}
...
var customer = new Customer();
customer.CalculateRating(); // this row will not work
```

        - protected
          - The type or member can be accessed only by code in the same class, or in a class that is derived from that class.

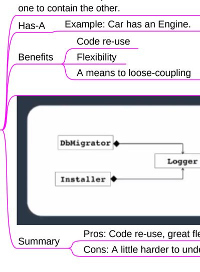
```
public class Customer
{
   protected int CalculateRating()
   {
   }
}
...
var customer = new Customer();
customer.CalculateRating(); // this row will not work
```

          - Try to avoid this
        - internal — The type or member can be accessed by any code in the same assembly, but not from another assembly.
        - protected internal — The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.
        - private protected — The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.
    - It's a good practice to create each new class in a different source file.
  - Coupling
    - A measure of how interconnected classes and subsystems are.
    - 

      - Tightly coupled (not recommended)
    - Loosely coupled (recommended)
    - To understand coupling
      - Encapsulation
      - The relationships between classes
        - Inheritance
          - A kind of relationships between two classes that allows one to inherit code from the other.
          - Is-A — Example: A Car is a Vehicle.
          - Benefits
            - Code re-use
            - Polymorphic behaviour
          - Syntax example

```
public class PresentationObject
{
   // Common shared code
}

public class Text : PresentationObject
{
   // Code specific to Text
}
```

          - Summary
            - Pros: Code re-use, easier to understand
            - Cons: Tightly coupled, fragile, can be abused
        - Composition
          - A kind of relationship between two classes that allows one to contain the other.
          - Has-A — Example: Car has an Engine.
          - Benefits
            - Code re-use
            - Flexibility
            - A means to loose-coupling
          - 

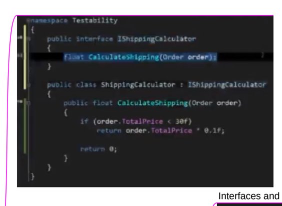
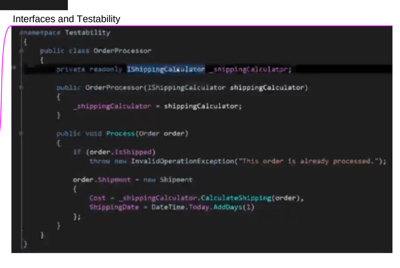
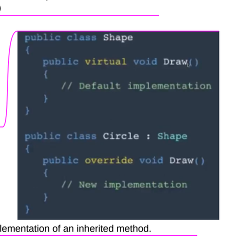
            - Syntax

```
public class Installer
{
   private Logger _logger;

   public Installer(Logger logger)
   {
      _logger = logger;
   }
}
```

          - Summary
            - Pros: Code re-use, great flexibility, loose coupling
            - Cons: A little harder to understand
        - Favour composition over inheritance
          - Problems with inheritance
            - Easily abused by amateur designers/developers
            - Large hierachies
            - Fragility
            - Tight coupling
          - Advantages of composition over inheritance — Any inheritance realtionship can be translated to composition.
          - Summary
            - 2 types of relationship between classes
              - Inheritance (Is-a)
              - Composition (Has-a)
            - Inheritance is not necessarily a bad thing
            - Use bot inheritance and composition
        - In UML
          - Parent/Base/Super class
          - Child/Derived class
        - Conversion
          - Conversion from a derived class to a base class (upcasting).
          - Conversion from a base class to a derived class (downcasting).
          - The as and is keywords.
      - Interfaces — A languae construct that is similat to a class (in terms of syntax) but is fundamentally different.

```
public interface ITaxCalculator
{
   int Calculate();
}
```

        - To build loosely-coupled applications.
          - 
          - Interfaces and Testability
          - Methodname_Condition_Expectation
          - 
  - Method
    - Main() — static public void Main(String[] args)
      - static: It means Main method can be called without an object.
      - public: It is access modifiers which means the compiler can execute this from anywhere.
      - void: The Main method doesn't return anything.
      - main(): It is the configured name of the Main method.
      - String[] args: For accepting the zero-indexed command line arguments. args is the user-defined name.
    - Method call
    - Method return code
      - If return value is 0 this means that the method returned correctly.
      - A method can also return a value to the caller.
    - Arguments and parameters (s 50).
      - A parameter is the special kind of variable that is defined in the method header and used inside that method to represent the value that was fed into the method call.
        - Ex sillyReturn (int i) there sillyReturn has a single parameter and has the identifier i.
        - Parameter passing by reference (s 55).
          - Ex addOneToRefParam (ref int i)
          - side effects.
        - Passing parameter values as out reference (s 56).
          - Ex. readPerson (out string name, out int age);
          - If I mark the parameter as out I must have given them a value for the program to compile.
      - An argument is the value that is supplied to the method when it is called.
        - Ex sillyReturn(99) there the argument is the value 99.
        - Named arguments (s 53).
          - x = readValue(low:25, high:100, prompt: "xcvbnn");
          - Optional parameters must be provided after all the required ones.
    - Design with methods (s 52).
      - It saves you writing the same code twice.
      - If a fault is found in the code you only have to fix it in one place.
      - Moving code around and creating methods is called refactoring.
    - <Access Specifier> <Return Type> <Method Name>(Parameter List) { Method Body }
      - **Access Specifier** − This determines the visibility of a variable or a method from another class.
      - **Return type** − A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is **void**.
      - **Method name** − Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
      - **Parameter list** − Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
      - **Method body** − This contains the set of instructions needed to complete the required activity.
    - When declaring methods, you must follow the sequence of its base elements – first type of the returned value, then method name and in the end list of parameters, surrounded by parenthesis ()
    - Overriding
      - Virtual/override
        - 

```
public class Shape
{
   public virtual void Draw()
   {
      // Default implementation
   }
}

public class Circle : Shape
{
   public override void Draw()
   {
      // New implementation
   }
}
```

      - Modifying the implementation of an inherited method.
    - Overloading
  - Block
    - Braces { }
    - A block of code starts with a { and ends with a }.
    - Nested blocks
      - Inner block
      - Outer block
  - Variables
    - A named location where you can store something.
    - The type of the variable is part of the metadata about variables.
    - Integers
    - Real
    - Char
      - Character Escape Sequences (s 25).
      - Unicode
    - String
      - Expressing text with no escape characters, putting a @ in front of the literal, ex @"\x0041CNSDE" .
      - Use " " around strings.
    - Bool
      - True
      - False
    - Control variables — i
    - C# does not let a variable in an inner block have the same name as one in an outer block.
    - Local variables disappear when the program execution leaves the block where they are declared.
    - Share a variable in two methods, you have to make the variable a member of the class.
    - A constant variable is given a value when it is declared.
  - Constants
    - It will never changed within the program.
    - There is a convenstion that you always give constant variables names which are expressed in CAPITAL_LETTERS with an underscore between words.
  - Arrays
    - An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

      Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

      All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.
    - datatype[] arrayName;
      - datatype is used to specify the type of elements in the array.
      - [ ] specifies the rank of the array. The rank specifies the size of the array.
      - arrayName specifies the name of the array.
    - An array allows us to declare a whole row of a particular kind of box.
    - Array elements
      - Elements — Each compartment in the box is called an element.
      - Subscripts
        - Indicate which box in the row that we want to use ex car[5]
        - Indexvärde på svenska.
      - Starting at 0.
      - When an array is created all the elements in the array are set to 0 (s 63).
    - Array size
      - Use of constant variables to hold the size of the array.
        - It makes the program easier to understand.
        - It makes program easier to change.
      - One dimension. — []
      - Two dimension.
        - The first subscription could be used to specify which row and the second which column.
        - [,]
      - Three dimensions. — [,,]
      - More than tree dimensions. — C# can handle this but is very hard to understand and visualise.
    - Risk if try to use an invalid value. We can guard against this by making that element into a null reference (s 65). — Ex monthNames[0] = null;
  - Collection
    - Listor
    - Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.

      Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. These classes create collections of objects of the Object class, which is the base class for all data types in C#.
      - It represents ordered collection of an object that can be **indexed** individually.
      - It is basically an alternative to an array. However, unlike array you can add and remove items from a list at a specified position using an **index** and the array resizes itself automatically. It also allows dynamic memory allocation, adding, searching and sorting items in the list.
    - List<string> strängLista = new List<string>();
      - Listan med strängar "strängLista" är en ny lista med strängar.
      - strängLista.Add("Mitt strängvärde");
      - Console.WriteLine(strängLista[0]);
      - Exempel 1: Användaren får skriva ett värde som sparas i listan i ett nytt element
        strängLista.Add(Console.ReadLine());
        Exempel 2: Användaren skriver in en siffra som vi sparar i en intlista
        Int32.TryParse(Console.ReadLine(), out int resultat);
        sifferLista.Add(resultat);
        Exempel 3: Skapar en vektor, ger dess element värden, och sparar vektorn i listan.
        string[] strängVektor = new string[2];
        strängVektor[0] = "Värde nummer 1";
        strängVektor[1] = "Värde nummer 2";
        vektorLista.Add(strängVektor);
      - Clear
        - strängLista.Clear();
        - Clear tar bort alla element i listan
      - RemoveAt
        - strängLista.RemoveAt(0);
        - RemoveAt kan användas för att ta bort elementet på ett specifikt index
      - IndexOutOfRange
      - Initiering
        - string[] strängVektor = new string[2] { "Första värdet", "Andra värdet" };
        - List<string> strängLista = new List<string>{ "Första", "Andra" };
  - Semicolon — Mark the end of the list of variable names and also the end that declaration statement.
  - Console
  - Parameters — A parameter is a mean of passing a value into a method call.
  - Components — Expose methods to get the job done.
  - Parse
    - Converting a string.
    - The process of looking along things is often called parsing.
  - Operator and operands (s 29)
    - Operator cause an operation to take place, ex + *
    - Operands are what the operators works on.
    - Operatorer (s 45)
      - Logiska — || &&
      - Tilldelning — = += -= /= *=
      - Jämförelse — == < > != >= <=
      - Unary operator
        - i++ — Means: Give me the value before the increment.
        - ++i — Means: Give me the value after the increment.
  - Program layout
    - It's more clear where things belong and make sense of what it does. In other words, easier to read and understand the program.
    - It should be easy to read.
    - It should have good punctuation and grammar.
    - It should look good on the page.
    - It should be clear who wrote it, and when it was last changed, and what's changed.
    - A big part of a well written program is the comments.
      - Line comments — //
      - Block comments — /* */
  - Declaration — A declaration identifies the type of the thing we want to store.
  - Expressions — An expression is something which can be evaluated to produce a result.
  - Overflowing — Wrap round — It's imported to know if you exceed the range of a variable.
  - File path
    - @ at the begininning of the literal means that doesn't contain control characters.
    - path = @"c:\data\2009\test.txt";
  - Stream — A stream is a link between your program and a data resource.
- Datatyper
  - Variabler_och_konstanter
    - const
      - Integer Literals
      - Floating-point Literals
      - Character Constants — There are certain characters in C# when they are preceded by a backslash.
        - Escape sequence
          - \\ Meaning \ character.
          - \" Meaning " character.
    - variabler — local
    - Containers
      - Vektorer (Arrays)
        - int[] sifferVektor = new int[5]; — Heltalsvektor "sifferVektor" är en ny vektor med 5 element.
        - string[] strängVektor = new string[4]; — Strängvektor med plats för 4 element
        - bool[] boolVektor = new bool[3]; — Boolvektor med plats för 3 element.
        - Indexvärde
          - Första elementet så måste vi peka på indexvärde 0.
          - Indexvärde räknas alltid från 0.
        - Felhantering
          - Till en början oinitierade dvs innehåller ett nullvärde.
          - Inkorrekta indexvärden. — IndexOutOfRange-fel.
          - TryCatch
      - Listor
  - Identifiers
    - Inga_reserverade_ord — @int
    - Ej_börja_med_siffra — oneRoute
    - Inga_mellanslag_i_namnet — firstName
    - Använd_meningsfulla_namn
      - Naming_Conventions
        - Hungarian Notation: strFirstName — Undvik_denna_i_C#
        - Pascal_Case: FirstName — **class names, method names, interface, public member/field variable, Namespaces**
        - Understykningstecken: _firstName — **Private member/field variable**
        - Camel_Case: firstName — **for method arguments and local variables**
  - Value types (Are stored on the stack).
    - Primitive Types
      - Integral Numbers
        - byte — 0-255
        - short — -32768-32767
        - int
          - int number;
          - int Number = 1;
          - -2100M-2100M
        - long
      - Real Numbers
        - float — const float Pi = 3.14f;
        - double
        - decimal — number = 1.2m;
      - Character — char — Unicode Characters
      - Boolean — bool — True/False
    - The struct type
  - Reference types (Are stored in the heap).
    - Non-Primitive Types
      - string
        - Console.WriteLine("Detta {1} är {0} första strängvärdet: {2} ----------> ", strängVektor[0], kalle[1], boolKalle[0]);
        - {0} {1} osv. beroende på antal parameterar som sätts in i texten.
      - array
      - enum
        - An enumeration is a set of named integer constants. An enumerated type is declared using the enum keyword.

          C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.
        - enum <enum_name> { enumeration list };
        - Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0. For example −

          enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
        - The enum_name specifies the enumeration type name.
        - The enumeration list is a comma-separated list of identifiers.

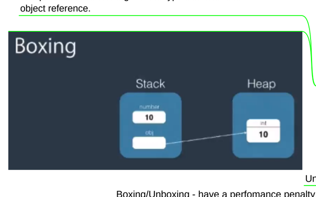
```
using System;

namespace EnumApplication {
   class EnumProgram {
      enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

      static void Main(string[] args) {
         int WeekdayStart = (int)Days.Mon;
         int WeekdayEnd = (int)Days.Fri;

         Console.WriteLine("Monday: {0}", WeekdayStart);
         Console.WriteLine("Friday: {0}", WeekdayEnd);
         Console.ReadKey();
      }
   }
}
```

          - When the above code is compiled and executed, it produces the following result −

            Monday: 1
            Friday: 5
      - class
        - The Object class is the base of all classes in .NET Framework.
        - Boxing
          - The process of converting a value type instance to an object reference.
          - 

        - Unboxing
        - Boxing/Unboxing - have a perfomance penalty.
- Loop_och_selektion
  - Loops
    - While — Så länge villkoret är uppfyllt
    - do-while — Testar först efter genomgång av kodblocket
    - For
      - for(int i=0; i < 10; i++)
      - intVektor.Length fungerar.
      - OBS! FOR-loop Length fungerar inte med lista utan man måste använda Count pga vi inte vet listans antal element.

```
using System;

namespace ArrayApplication {
   class MyArray {
      static void Main(string[] args) {
         int [] n = new int[10]; /* n is an array of 10 integers */
         int i,j;

         /* initialize elements of array n */
         for ( i = 0; i < 10; i++ ) {
            n[ i ] = i + 100;
         }

         /* output each array element's value */
         for (j = 0; j < 10; j++ ) {
            Console.WriteLine("Element[{0}] = {1}", j, n[j]);
         }
         Console.ReadKey();
      }
   }
}
```

        - Element[0] = 100
          Element[1] = 101
          Element[2] = 102
          Element[3] = 103
          Element[4] = 104
          Element[5] = 105
          Element[6] = 106
          Element[7] = 107
          Element[8] = 108
          Element[9] = 109
    - Foreach
      - foreach (variabeltyp identifier in Container)
      - foreach(string minStrang in strangLista) — Det här fungerar på både vektorer och listor, så länge vi pekar på rätt sorts variabeltyp. Så har vi en stränglista måste vi säga att variabeltypen är string, är det en int-vektor måste vi säga att variabeltypen är int. Sen anger vi en identifier. Det här kan vara vad som helst, är det bara ett behändigt namn. Vi använder nyckelordet "in" för att påpeka att vi plockar den här variabeln ut ur Containern. Till sist anger vi vilken Container vi vill utgå ifrån.

```
using System;

namespace ArrayApplication {
   class MyArray {
      static void Main(string[] args) {
         int [] n = new int[10]; /* n is an array of 10 integers */

         /* initialize elements of array n */
         for ( int i = 0; i < 10; i++ ) {
            n[i] = i + 100;
         }

         /* output each array element's value */
         foreach (int j in n ) {
            int i = j-100;
            Console.WriteLine("Element[{0}] = {1}", i, j);
         }
         Console.ReadKey();
      }
   }
}
```

        - Element[0] = 100
          Element[1] = 101
          Element[2] = 102
          Element[3] = 103
          Element[4] = 104
          Element[5] = 105
          Element[6] = 106
          Element[7] = 107
          Element[8] = 108
          Element[9] = 109
    - Nested — Ibland kan det vara användbart att ha loopar innanför andra loopar, det kallas nästlade loopar. Om vi till exempel har en lista som innehåller vektorer så kan vi behöva en foreach-loop för att först plocka fram alla vektorer i listan, och sen en till foreach-loop för att plocka fram det som finns i vektorn. Vi tar ett exempel för att illustrera;

```
string[] strängVektor = new string[] { "Ett", "Två", "Tre" };
string[] andraVektor = new string[] { "Första", "Andra", "Tredje" };
List<string[]> vektorLista = new List<string[]>();
vektorLista.Add(strängVektor);
vektorLista.Add(andraVektor);
foreach( string[] vektor in vektorLista )
{
foreach( string varjeSträng in vektor )
{
Console.WriteLine(varjeSträng);
}
}
```

  - Selektioner
    - IF
      - else if
      - else
    - Switch
      - case 1:
        - break;
        - default
      - Adwise: Don't putting large amount of program code into a switch case. Instead you should put a call to a method.
- Metoder
  - static
  - Returvärde
    - void
    - int
    - string
  - Metodnamn(parameters)
    - Parameters
      - Arguments
      - int siffra1, int siffra2
      - string text1, string text2
      - Namngivna_parameters — Metodnamn(siffra! : 10, multiplikator : 5, siffra2 : 2)
      - Alternativa_parameters — **Addera(int siffra1 = 0, int siffra2 = 0, int siffra3 = 0, int siffra4 = 0, int siffra5 = 0)**
- Klasser
  - Statisk — Objekt
  - Icke statisk — Objekt
- Technical_stuff about C#
  - Managed and/or unmanaged parts
  - Object oriented language
  - Platform independent
  - Based on C++
  - C# is both interpreted and compiled in its lifetime
    - Compiled to a virtual language
    - Interpreted by a VM
  - C# is case sensitive.
  - All statements and expression must end with a semicolon (;).
  - The program execution starts at the Main method.
  - Unlike Java, program file name could be different from the class name.
- Felhantering
  - Parse — A method we use to convert strings of text into numeric values.
  - TryCatch — try - catch - finally
  - TryParse — Lämpligt vid konvertering av siffror
    - TryParse(int)
    - TryParse(bool)
  - Exception
    - An exception is a type of object that contains details of a problem that has occurred. — e.Message
    - Exception Nesting
    - Catching exceptions - "dynamic error handling"
  - Break_Points + Step_Into
  - Olika_typer_av fel
    - Kompileringsfel
    - Körtidsfel
    - Logiska_fel
- C#_Programming_Yellow_Book
  - Computers_and_Programs
    - A computer program is just a sequence of instructions which tell a computer what to do with the data coming in and what form the data sent out will have.
    - What information flows into the system?
    - What flows out of the system?
    - What the system does with the information?
    - Functional Design Specification, FDS
    - You do not work for your customers, you work with them.
    - A big part of learning to program is learning how to use all the additional features of the system which support your programs.
  - Simple_Data_Processing
  - Creating_Programs
  - Creating_Solutions
  - Advanced_Programming
  - Glossary_of_Terms
  - Gratisversion att ladda ned: — http://www.csharpcourse.com/
- Tools
  - Microsoft Visual Studio
  - Microsoft .NET Framework
  - C# Compiler