Web Systems Fundamentals 7.5 credits

A pragmatic and basic introduction to ASP.NET

- data types and variables
- methods and classes
- masterpages

HTML/CSS compared to ASP.NET

- **HTML and CSS**
  - *HyperText Markup Language / Cascading Style Sheets*
  - Extensions: html & css
  - Presentation and layout
  - Client-side (the code is parsed by the browser)

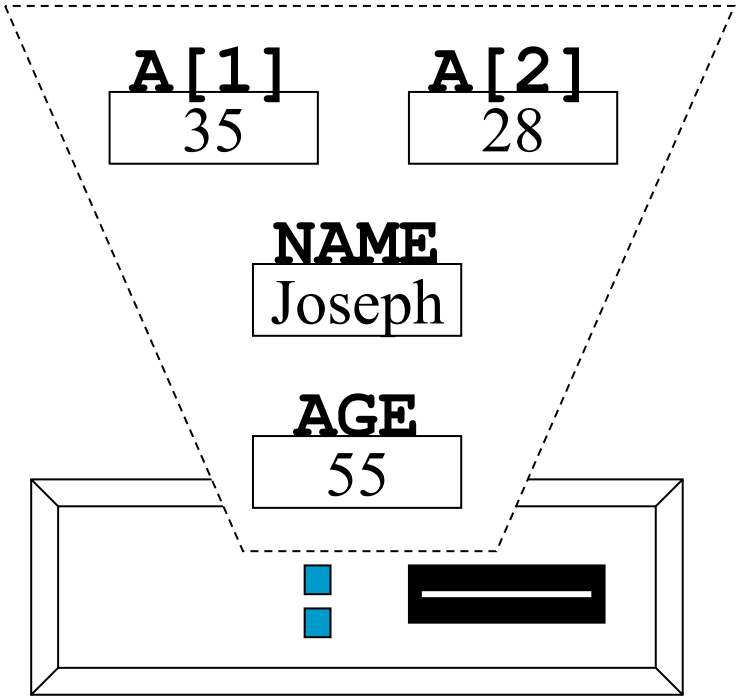
- **ASP.NET and C# (VB.NET etc.)**
  - *Active Server Pages*
  - Extensions: aspx & cs
  - Provides dynamics
  - Server-side (the code is parsed by the server)
    - Browser sends request to web server (via HTTP)
    - Web server sends a response to the browser

Common file extensions

- .html — static HTML pages w/o ASP.NET dynamics
- .css — stylesheet files (CSS code)
- .aspx — Web Forms (common pages)
- .ascx — Web User Control (reusable page fragments)
- .master — MasterPages (providing a global structure)
- .config — Web Configuration Files
- .sitemap — Hierarchical representation of files in XML (good for SEO)
- .cs — C# code files (commonly used by code behind files)
- .js — Javascript code files
- .xml — XML-file (eXtensible Markup Language)

Data types and variables

- A variable…
  - …provides temporary storage in RAM
  - …has a unique name (name, age)
  - …contains data (Joseph, 55)
  - …has a data type
  - …can be arranged in vectors/arrays that contain several variables of the same kind



- **Why?**
  - to store data temporarily while processing it
  - Analogy: a car's fuel tank
    - Name: fuel tank
    - Data type: gasoline

Commonly used data types

- byte
  - whole numbers 0-255
- int
  - Whole numbers between ± 2.147.483.647
- decimal
  - With up to 29 significant digits, most accurate for floating-point numbers
- double
  - as decimal but with a narrower range
  - faster, often preferred
- Float (called single in VB.NET)
  - as double but with a narrower range
- Bool (condition)
  - true or false
- char
  - A single character
- string
  - text strings (almost anything)
  - Up to 2 billion characters
- system.datetime
  - hh:mm:ss MM:DD:YYYY

Declaring variables (C#)

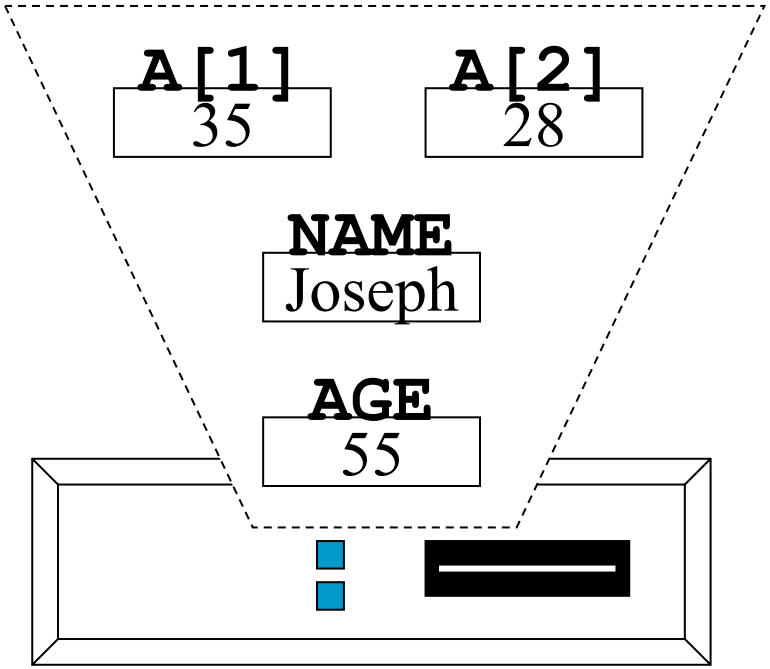
- Allocating space for a variable is called declaring

```
Datatype myVariable
```

- Ensure that variable names are easy to understand (and kept reasonably short)

```
// delcare a variable for storing text
string name;
int age;

// assigning a value
name = "Joseph";
age = 55;
```



Techniques used for naming variables

- Hungarian notation (prior to .NET)

```
string strName;
int intAge;
```

  - Considered obsolete due to IntelliSense in the Visual Studio IDE
  - Provides clear information, but is also harder to read

- Use **lowerCamelCase** for variables
- Assign them **useful and relevant names** which are easily understood

- **C# is case sensitive**:
  - squeakyBeavers isn't the same as squeakybeavers

Converting data types

Occasionally data in variables need to be converted before processing

(use ”data” correctly: /ˈdeɪtə/ - plural of datum!)

- The .ToString method (convert almost anything to a string)
  - `label1.text = System.DateTime.Now.ToString();`
- Convert
  - `bool myBool = Convert.ToBoolean(”True”);`
- Casting (using brackets before the variable)
  - only works with compatible types, e.g. double to int, but not datetime to int)
  - ```
    object o1 = 1;
    int i1 = (int)o1;
    ```
    ```

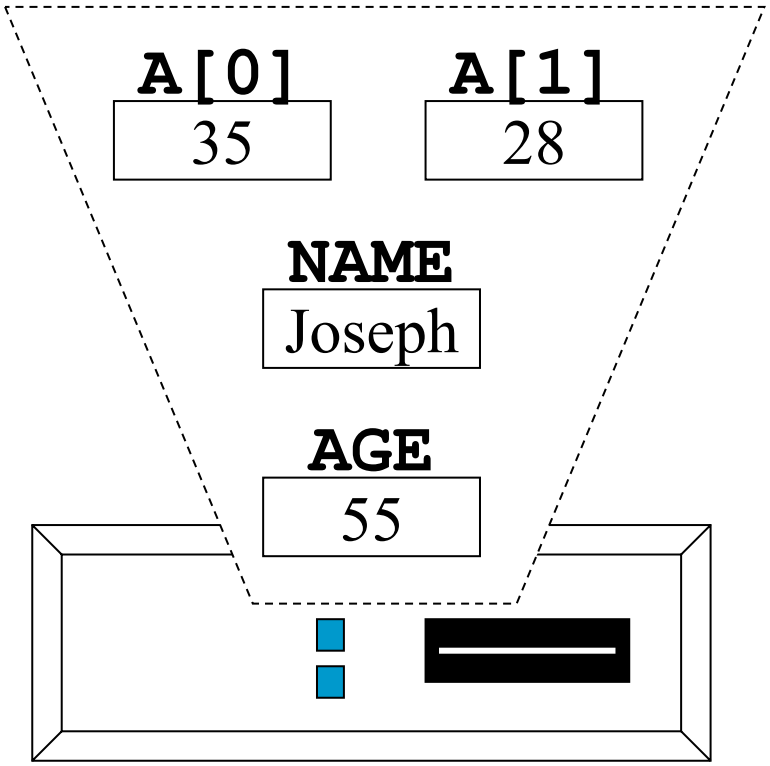
Arrays: bags with pockets

Data structures are used to store a large amount of data in the same place

- Declaration of an array (square brackets)

```
/* declare an array with the size of two
fields, called a[0] och a[1] */
Int[] a = new int[2];
```

- Assignation
  - `a[0] = 35;`
  - `a[1] = 28;`
- The size is pre-defined
  - `a[2] = 56; // fail`
- Resizing
  - `Array.Resize<int>(ref a, 3);`
  - `a[2] = 56; // works after resizing`



Collections: bags without pockets

- Similar to arrays, but slightly more flexible
- Enables storage of more than one object in a variable

- ArrayLists inlclude properties like add, remove och clear
- Accepts all objects – irrespective of the data type of the object = problematic

```
// create an arraylist
ArrayList peopleInClass = new ArrayList();
peopleInClass.Add("Thomas DiLeva"); //add
peopleInClass.Add("Justin Beaver"); //add
peopleInClass.Remove("Justin Beaver"); //remove
peopleInClass.Clear(); //clear the list
```

Collections (part II)

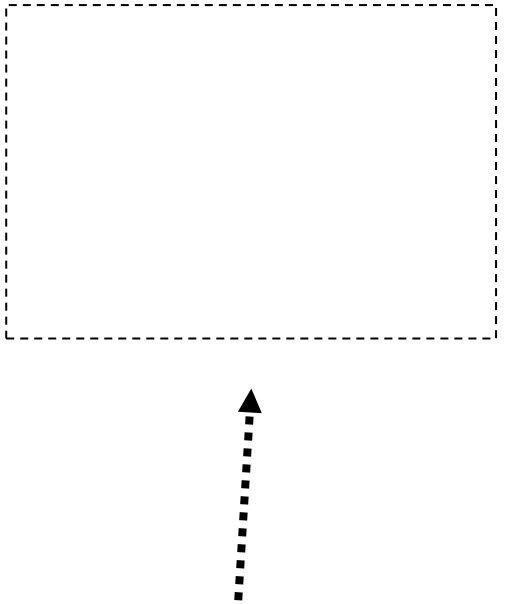
- ArrayList
  - General collection for objects
- HashTable
  - Storing key/value pairs
  - Keys must be unique
- Queue
  - First-in, first-out collection
- Stack
  - Last-in, first-out collection
- SortedList
  - Ordered storage for key/value pairs

Choosing between the two

- **Use arrays** when the number of elements don’t change
  - Faster and more efficient
- **Use collections** when you need dynamic allocation of elements
  - and special functionalities (search, sort, etc)

Working with nothing at all

- The value 0 (zero) is considered a value
- A variable without a value has a status called null
- Could be problematic



Operators

- Easy to understand
  - + - * /
- A little more advanced
  - Exponentiation ^
    - 2^3 = 2 * 2 * 2 = 8
      Math.Pow(2, 3); // 8 (C#)
  - Modulus (remainder)
    - 1 % 2 = 1 // rest 1
    - 2 % 2 = 0 // ingen rest

- Comparison operators
  - Used in flow control
    - ==, !=, <, >, <=, >=
- Logical operators
  - Allows combinations of expressions
    - & (And) | (Or) ! (Not)
  - Special cases
    - && (AndAlso) || (OrElse)

Operator precedence

- Calculations are made from left to right, and with operator priority (if not changed by the use of a parenthesis)
  - 1 + 2 * 3 = 7, but (1 + 2) * 3 = 9

- Operator priority

  * / % + -

Using strings

- A string is a sequence of characters
- Text boxes produce strings
- Strings are easy to manipulate
- Keep in mind that a blank (space) also is a character

- Comparation

```
If (name1 == name2)
{// code that performs something
}
```

- Concatenation

```
string firstPart = ”Where did my”;
string secondPart = ”dignity go?”;
string result;
result = firstPart + secondPart;
```

Three things to remember

- Originally a part of Jackson Structured Programming (JSP) - all applications can be described with three processes
  - **Sequence: do in order**
    - Do first A, then do B
  - **Selection: making choices**
    - Condition determined action
  - **Iteration: something to repeat**
    - Repeat until terminated

Selections

- If…Then, run on True

```
if (weather==”good”)
{
      skipClass=true;
}
```

Including Else

```
if (weather==”good”)
{
      skipClass=true;
}
else
{
      skipClass=false;
}
```

- Multiple testing

```
if (user.IsInRole(”User”)
{
      deletebutton.visible=true;
}
else if user.IsInRole(”Admin”)
{
      deletebutton.visible=true;
}
else
{
      deletebutton.visible=false;
}
```

- Alternative methods for selection exists

switch

Switch

- If multiple alternatives exist, an alternative selection method is the switch

```
switch (lstOperator.selectedValue)
{
Case "+";
   result=value1+value2;
   break;
Case "-";
   result=value1-value2;
   break;
}
resultlabel.text=result.ToString();
```

- Multiple testing
  - Single test
    `Case 3`
  - Multiple test
    `Case 3, 5, 8`
  - Range test
    `Case 6 To 11`
  - Logic test
    `Case Is <25`
  - Boolean test
    `Case True`

Iterations (loops)

- While statements test condition at start

```
while (count < 6)
{
   textbox.text = count;
   count++;
}
```

- Do loops

```
do {
    textbox.text = count;
   count++;
} while (count < 6);
```

FOR – NEXT iterations

- Used to repeat actions
  - General form

```
for (int i = 1; i <= 5; i++)
{
  textbox.text = i;
}
```

FOR - EACH iterations

- Loops through arrays and collections

```
foreach (int i in myArray)
   {
     label.text = i;
   }
```

- Very useful for the Collection object, since you don’t need to know how many entries there are in the collection

Object Oriented Programming (OOP)

C++

Objective C

Java

Ruby

C#

VB.NET

Advantages with OOP

- **Modularity**
  - code is devided into classes
  - grouping of similar functionality
- **Extensibility**
  - Classes are extensible
  - You don’t have to copy and redo everything
- **Reusability**
  - You can build libraries of code which can be used over and over again just by referring to it

What is an object?

The TV as a metaphor:

The TV is an **object** which is able to **do** **stuff** = The TV **has got** **properties**

Switch channel
Change volume
Change status (on/off)

When you want to do something, you **invoke/call** the object

What is a class?

A collection of code used to control methods and events
OOP is basically about dividing code into classes

- **An object is a copy of a Class**.

Using the TV-metaphor again:

- If you have a TV in your livingroom and a TV in the bedroom, you have **two objects**, but they come **from the same class** and have the **same properties**.

Create vs instantiate

You **create** a class, but you **instantiate** an object

Using the car as an example

- You **design** a car (decide specifications etc) = **create** class
- You **manufacture the** car in order to drive it= **instantiate** object

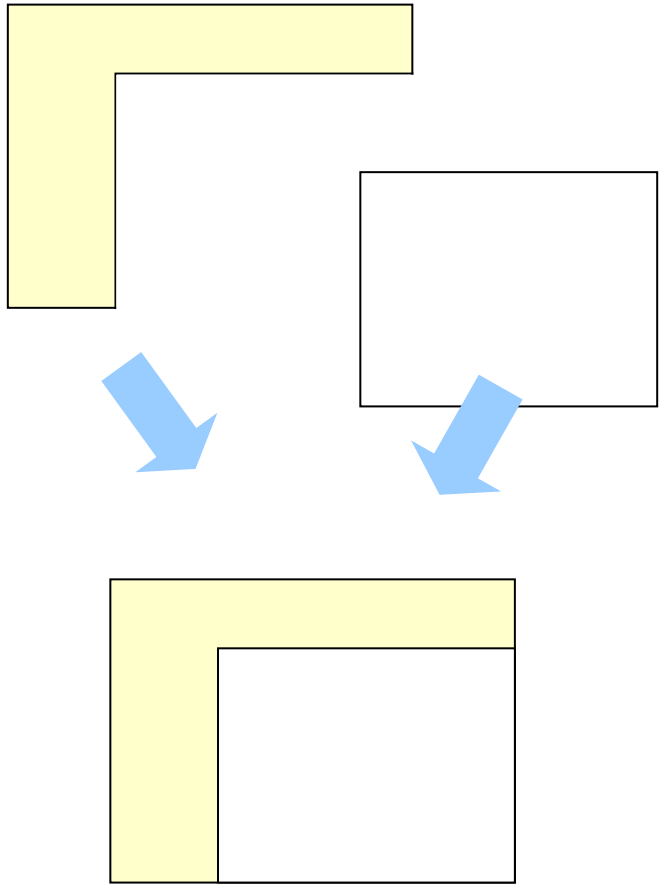
Classes and methods

```
Public Class Employee //class
{
  Public void Work() //method
  {
       // code that performs something goes here
  }
}
```

Methods include code that performs an action

Master and content pages

- A master page is a template (mainly for layout purposes) – yellow
- A content page is a dynamic part of a page which changes – white
- Thanks to masterpages, you can make a universal layout for your entire website, without having to copy the code all over again on every page



Master and content pages

- Start the Masterpage (the template) with:

```
<%@ Master Language=”C#” %>
```

- Start the other pages with:

```
<%@ Page Title=”” Language=”C#”
MasterPageFile=”~/MasterPages/Name.master”
AutoEventWireup=”true” CodeFile=”Default.aspx.cs”
Inherits=”_Default”>
```

- Placeholder for content between the HTML body tags:

```
<asp: contentplaceholder id=”contentName” runat=”server”>
        // put your content here
</asp: contentplaceholder>
```