**Samenvatting Software Development**

Table of Contents

[Data types & variables 2](#_Toc180055691)

[String 2](#_Toc180055692)

[Enums 3](#_Toc180055693)

[Casting 3](#_Toc180055694)

[Implicit Casting 3](#_Toc180055695)

[Explicit Casting 3](#_Toc180055696)

[Conditionals 4](#_Toc180055697)

[if statements 4](#_Toc180055698)

[switch block 5](#_Toc180055699)

[Ternary Operator 5](#_Toc180055700)

[Math 6](#_Toc180055701)

[Loops 7](#_Toc180055702)

[While loop 7](#_Toc180055703)

[For loop 7](#_Toc180055704)

[Foreach loop 7](#_Toc180055705)

[Do-While loop 8](#_Toc180055706)

[Break and Continue 8](#_Toc180055707)

[Methods 9](#_Toc180055708)

[Lists 10](#_Toc180055709)

[Classes 11](#_Toc180055710)

# Data types & variables

|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Size | Assignment | Description |
| int | 4 bytes | int a = 5; | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | long a = 54234; | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | 4 bytes | float a = 5.452F; | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | double a = 5.423D; | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| decimal | 16 bytes | decimal a = 5.95M; | Stores fractional numbers. Sufficient for storing 28-29 significant digits |
| bool | 1 bit | bool a = true; | Stores true or false values |
| char | 2 bytes | char a = ‘a’ | Stores a single character/letter, surrounded by single quotes |
| string | 2 bytes per character | string a = “Hello”; | Stores a sequence of characters, surrounded by double quotes |

## String

A string in C# is actually an object, which contain properties and methods that can perform certain operations on strings. For example, the length of a string can be found with the Length property:

string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
Console.WriteLine(txt.Length)

There are many string methods available, for example ToUpper() and ToLower(), which returns a copy of the string converted to uppercase or lowercase:

string txt = "Hello World";  
Console.WriteLine(txt.ToUpper()); // Outputs "HELLO WORLD"  
Console.WriteLine(txt.ToLower()); // Outputs "hello world"

## Enums

An enum is a special "class" that represents a group of **constants** (unchangeable/read-only variables).

To create an enum, use the enum keyword (instead of class or interface), and separate the enum items with a comma:  
enum Months

{

January, // 0

February, // 1

March, // 2

April, // 3

May, // 4

June, // 5

July // 6

}

## Casting

In C#, there are two types of casting:

* **Implicit Casting** (automatically) - converting a smaller type to a larger type size  
  char -> int -> long -> float -> double
* **Explicit Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char

### Implicit Casting

int myInt = 9;  
double myDouble = myInt;

### Explicit Casting

double myDouble = 9.78;  
int myInt = (int)myDouble;

# Conditionals

C# supports the usual logical conditions from mathematics:

* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b
* Equal to a == b
* Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

C# has the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed

## if statements

if (*condition1*)  
{  
 *// block of code to be executed if condition1 is True*  
}   
else if (*condition2*)   
{  
 *// block of code to be executed if the condition1 is false*

*and condition2 is True*  
}   
else  
{  
 *// block of code to be executed if the condition1 is false and condition2 is False*  
}

## switch block

switch(*expression*)

{  
 case x:  
 *// code block*  
 break;  
 case y:  
 *// code block*  
 break;  
 default:  
 *// code block*  
 break;  
}

## Ternary Operator

variable = (condition) ? expressionTrue : expressionFalse;

Console.WriteLine(“De student heeft” + (cijfer > 8) ? “Cum laude” : “GEEN Cum laude”)

# Math

The C# Math class has many methods that allows you to perform mathematical tasks on numbers.

The Math.Max(*x*,*y*) method can be used to find the highest value of *x* and *y*:  
Math.Max(5, 10);

The Math.Min(*x*,*y*) method can be used to find the lowest value of of *x* and *y*:  
Math.Min(5, 10);

The Math.Sqrt(*x*) method returns the square root of *x*:  
Math.Sqrt(64);

The Math.Abs(*x*) method returns the absolute (positive) value of *x*:  
Math.Abs(-4.7);

Math.Round() rounds a number to the nearest whole number:  
Math.Round(9.99);

# Loops

## While loop

The while loop loops through a block of code as long as a specified condition is True:  
while (*condition*)   
{  
 *// code block to be executed*  
}

**Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!**

## For loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:  
for (int i = 0; i < 5; i++)   
{  
 Console.WriteLine(i);  
}

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

## Foreach loop

There is also a foreach loop, which is used exclusively to loop through elements in an **array** or a **list** (or other data sets):

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
foreach (var car in cars)   
{  
 Console.WriteLine(i);  
}

## Do-While loop

The while loop loops through a block of code as long as a specified condition is True but always runs ONCE:  
do   
{  
 *// code block to be executed*  
} while (*condition*)

## Break and Continue

The break statement can be used to jump out of a **loop**.  
int i = 0;  
while (i < 10)   
{  
 Console.WriteLine(i);  
 i++;  
 if (i == 4)   
 {  
 break;  
 }  
}

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.  
int i = 0;

while (i < 10)

{

if (i == 4)

{

i++;

continue;

}

Console.WriteLine(i);

i++;

}

# Methods

A method is defined with the name of the method, followed by parentheses **()**. C# provides some pre-defined methods, which you already are familiar with, such as Main(), but you can also create your own methods to perform certain actions.

* MyMethod() is the name of the method.
* static means that the method belongs to the Program class and not an object of the Program class.
* void means that this method does not have a return value. You can use all datatypes as return values.

To call (execute) a method, write the method's name followed by two parentheses () and a semicolon;

static void MyMethod()

{

Console.WriteLine("I just got executed!");

}

static void Main(string[] args)

{

MyMethod();

}

// Outputs "I just got executed!"

When the function needs data parsed to is, you need to include that variable(s) in the parentheses!

static void MyMethod(string fname)

{

Console.WriteLine(fname + " Refsnes");

}

static void Main(string[] args)

{

MyMethod("Liam");

MyMethod("Jenny");

MyMethod("Anja");

}

// Liam Refsnes

// Jenny Refsnes

// Anja Refsnes

# Classes

Everything in C# is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects / records.

public static class Rekenmachine

{

  public static decimal Optellen(int input1, int input2)

  {

    return input1 + input2;

  }

  public static decimal Aftrekken(int input1, int input2)

  {

    return input1 - input2;

  }

  public static decimal Vermenigvuldigen(int input1, int input2)

  {

    return input1 \* input2;

  }

  public static decimal Delen(int input1, int input2)

  {

    return input1 / input2;

  }

}