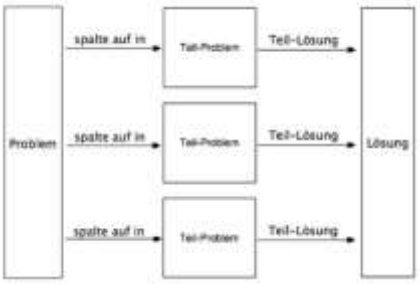
Modul 403

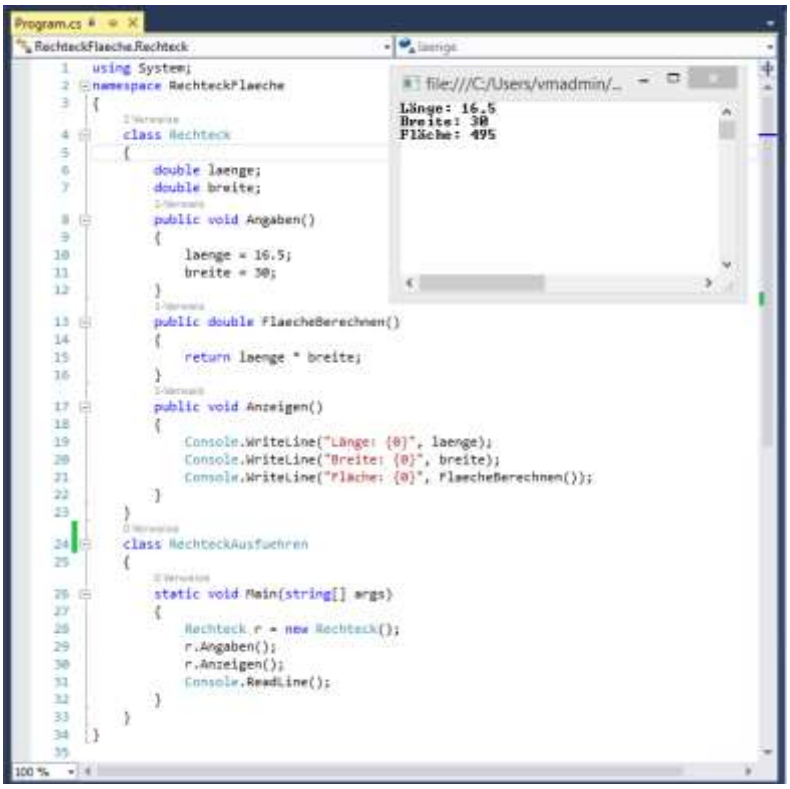
Natürliche Sprache -> Wird von Menschen gesprochen und es ist eine Vereinbarung im Sinne der Kommunikation. Menschen können Inhalte auch verstehen wenn die Sätze falsch formuliert werden.



Programmiersprache -> Mensch-Computer-Schnittstelle. Es ist ein formales Regelwerk mit definierter Syntax. Es gibt keine Toleranz bei Fehlern. **Die Grammatikregeln der Syntax stehen den Interpretationsregeln der Semantik gegenüber.**

Der Wortschatz ist wesentlich kleiner und kann sich mit den Versionen immer wieder ändern und natürliche Sprachen haben auch verschiedene Wörter mit der gleichen Bedeutung, die Programmiersprachen nicht haben. Syntax Überprüfung mit **Auto allicence.**

Prozedurale Programmierung -> Programmierparadigma(Programmier-Stilrichtung) - Ursprung 1960. Überarbeitung der strukturierten Programmierung. -> Problem nachhaltig lösen, das heisst der Programmcode soll später noch weiterverwendet werden können. In Teil Probleme aufteilen. Mit **Prozeduren und Funktionen** nachhaltig lösen. **Prozedur** liefert im Gegensatz zur Funktion kein Resultat.



Objektorientierung: Methoden und Konzepte zur Strukturierung von Programmcode für komplexe Anwendungen. Daten und zugehörige Funktionen werden in Klassen von Realobjekten zusammengefasst. Wiederverwendbarkeit Code, bessere Lesbarkeit und Verständlichkeit und einfache Wartbarkeit.

Nicholas Wirth -> Wichtige Persönlichkeit CH Inf und Inf ETH Professor -> genervt von der Komplexität der Progsprachen schuf er Pascal, Modulo /-2 und Oberon.

Visual C#Sharp von Microsoft -> auf bmwp2 berücksichtigt alle modernen objektorientierten Konzepte sowie Methoden und ist State of the art.

Satz des Pythagoras in der formalen Syntax
c=Wurzel(Summe(quadrat(a), quadrat(b)))

Funktion mit sprechendem Namen:

1. wurzel(x)= wurzel x
2. summe(x+y) =x+y
3. quadrat(x)=x^2

Befehle:

using besagt, dass der System-Namensraum in das Programm aufgenommen wird. Ein Programm hat in der Regel mehrere using-Statements.

namespace Begriff -objektorientierten Programmierung – die Namen für Objekte werden in einer Baumstruktur angeordnet und über entsprechende Pfadnamen eindeutig angesprochen. **Vereinfacht bedeutet dies, dass innerhalb eines solchen Raumes jeder Name eindeutig ein Objekt bezeichnet äusserste Klammer eines Programms**. Die Bezeichnung oder auch Identifier kann frei gewählt werden nach namespace ...ausser keywords von C#

class ist ein Bauplan aus der dann ein reales **Objekt** erzeugt werden kann. In der Regel beinhalten Klassen **Daten (Eigenschaften) und Operationen (Methoden)** die auf den Daten operieren. Methoden definieren das Verhalten und die Eigenschaften den Zustand der Klasse.

Main Ist eine Methode, die automatisch aufgerufen wird, auch Einstiegs- oder Startpunkt des Programms.

void (in Main gibt es nur int oder void) = leer ohne rückgabewert **static** = nicht veränderbar -> **Static** oder **Public** sind **Zugriffsmethoden**

static void Main(string[] args) **-> string[] args =** Indexiertes Array (nummerierte Liste)

Anderes als Main: Könnte auch **public double Flachenberechnung()** heissen

Kommentare // für eine Zeile /*...*/ für ganze Abschnitte

WriteLine();In der **Main Methode** steht das Statement, welches „Hello World“ ausgibt: Console**.WriteLine**("Hello World");
Die Methode WriteLine aus der Klasse Console, die im Namensraum System definiert ist, gibt nun „Hello World“ aus.

Console**.ReadKey();** ist für die VS Benutzer/innen. Das Programm wartet auf einen Tastendruck, um beendet zu werden. Sonst nur mili sekunden zu sehen.

Variable = abstrakte Grösse mit Variablennamen und Adresse im Arbeitsspeicher -> Bestandteile Datentyp Variablenname; = Datenwert;

Alle Befehle enden mit „;“

Namensräume, Klassendeklarationen und Methoden haben nach der Bestimmung des Namens kein Semikolon „;“, umklammern aber die Definition mit geschweiften Klammer {...}

C# ist case sensitiv = Es achtet auf Gross / Kleinschreibung

Instanziierung: Rechteck r = new Rechteck(); -> generieren eines Objekts nach dem Bauplan class Rechteck vom Datentyp Rechteck mit dem Namen r.

Konstruktor oder Destruktor sind Methodenaufrufe z.B. Rechteck() oder ReadLine() Konstruktor mit Parametern versehen und Destruktor nicht.

Variablendeklaration: Datentyp Variablenname1, Variablenname2; **oder** Datentyp Variablenname = Datenwert;

2. Datentypen(Wertetypen):

Double(float, decimal) = Fliesskommazahlen

Integer(Int32/64)(long,sbyte,short,uint,ulong,uahort) = Ganze Zahlen

Boolean = Wahrheitswert True oder False

String = Text

Grösse von Wertetypen = sizeof(z.B. double)

3. Datentypen(Verweistypen/Referenztyp): Array, String, Klassen, Delgaten (**Verweistyp** wird mit **new** deklariert sie werden langfristig gespeichert im heap, später wieder verwendbar) -> z.B. Rechteck r = new Rechteck();

Wertetypen in Verweistypen verwandeln: Boxing -> object name = variablezumboxen; **Unboxing ->** Wertetyp Neuevariable = (Wertetyp)name;

4. Kontrollstrukturen: Entscheidung ob Bedingung ausführen oder nicht

IF Anweisung: führt etwas nur bei einer gewissen Bedingung aus: **if (variable == „“)** ohne {} und wird als True False interpretiert optional noch mit **else**

else wird immer dem nächstliegenden if zugeordnet, dies muss durch {} def werden.

man darf das 2. If auch direkt nach dem else schreiben.

Vergleichsoperatoren = ><= is as == != & && |(or):

Switch Anweisung: Wenn mehr als 2 alternativen (**nur ganzzahlige Datentypen, Zeichnenketten**., sodass man unter umständen trotzdem auf if zurückgreiffen muss)

```
switch(variable)
{
   case „Hallo“:
      Console.WriteLine(„“);
      break; //Muss immer geschrieben werden! Sonst meldet fehler
   default: //Optional
      Console.WriteLine(„“);
      break;
}
```

Keywords C#	
abstract	namespace
as	new
base	null
bool	object
break	operator
byte	out
case	override
catch	params
char	private
checked	protected
class	public
const	readonly
continue	ref
decimal	return
default	sbyte
delegate	sealed
do	short
double	sizeof
else	stackalloc
enum	static
event	string
explicit	struct
extern	switch
false	this
finally	throw
fixed	true
float	try
for	typeof
foreach	uint
goto	ulong
if	unchecked
implicit	unsafe
in	ushort
int	using
interface	virtual
internal	void
is	volatile
lock	while
long	

Begriffe:

Code Highlighting oder **Syntax Highlighting** = Ist eine Farbliche Unterliegung (z.B. Klassenbezeichnung = blau)

Array = mehrere Datentypen in Liste -> PrintArray(Variable[Nummer]); **string[] args =** Indexiertes Array

Konstruktor(Methodenaufruf) = Rechteck()

Eingabe Konvertieren: `a(zuerst definieren) = Convert.ToDouble(Console.ReadLine());`

Methoden=Convert ToString,ToInt64,ToInt32 **??** ToLower,ToUpper,**Console** Write, WriteLine, ReadKey,ReadLine

Multiplicative	* / %
Additive	+ -
Shift	<< >>
Relational and type testing	< > <= >= is as
Equality	== !=
Logical	AND &
Logical	XOR ^
Logical	OR |
Conditional	AND &&
Conditional	OR ||
Conditional	?:
Assignment	= *= /= %= += -= <<= >>= &= ^= |=

```
// MODUL 403: Programmabläufe prozedural implementieren
// M403-Codebeispiel-9:  elseif-Anweisung
// Ralph.Maurer@gmx.ch
// http://www.iet-gibb.ch
using System;
public class Entscheidung
{
  public static void Main()
  {
    string Name;
    Console.Write("Bitte geben Sie Ihren Namen ein: ");
    Name = Console.ReadLine();
    if(Name == "Bill Gates")
    {
      Console.WriteLine("Sie arbeiten wohl bei Micosoft!");
      Console.ReadLine();
    }
    else if(Name == "Steve Jobs")
    {
      Console.WriteLine("Danke für den Mac, den iPod, das iPhone, das iPad und die
Vision es besser zu machen.");
      Console.ReadLine();
    }
    else if(Name == "Niklaus Wirth")
    {
      Console.WriteLine("Prozedural - können wir alle mal!");
      Console.ReadLine();
    }
    else {
      Console.WriteLine("Tut mir leid, aber Sie kenne ich nicht.");
      Console.ReadLine();
    }
  }
}
```

Funktion = Wenn Funktion() muss ein return... ; in dem Abschnitt vorhanden sein

```
using System;
namespace Pythagoras
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("---------------------------------");
            Console.WriteLine("Pythagoras: c^2 = a^2 + b^2");
            Console.WriteLine("---------------------------------");

            double c, a, b, quadrat_a, quadrat_b, quadratsumme;

            Console.WriteLine("Geben Sie einen Wert für a ein:");
            a = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Geben Sie einen Wert für b ein:");
            b = Convert.ToDouble(Console.ReadLine());

            quadrat_a = a * a;
            Console.WriteLine("a im Quadrat = {0}", quadrat_a);

            quadrat_b = b * b;
            Console.WriteLine("b im Quadrat = {0}", quadrat_b);

            quadratsumme = quadrat_a + quadrat_b;
            Console.WriteLine("Quadratsumme von a und b = {0}", quadratsumme);

            c = Math.Sqrt(quadratsumme);
            Console.WriteLine("Hypotenuse c = {0}", c);

            Console.ReadLine();
        }
    }
}
```