SNABBGUIDE STRUKTURER I C#

Dessa paket måste alltid finnas (om utmatning och inmatning ska fungera)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

Denna function måste alltid finnas

```
static void Main(string[] args)
{
    …
    Console.ReadKey();
}
```

Variabler som används måste deklareras t.ex.

```
int heltal, antal, nr ;
double decimaltal, tal1, tal2, tal3 ;
char c;
string str =”Programmera C#”;
```

Ibland måste vi initiera variablerna för att det inte ska bli fel t.ex.

```
int summa = 0, produkten = 1 ;
double domarsumma = 0.00 ;
```

Utmatning på skärm med avslutande radbrytning skrivs

```
Console.WriteLine("text");
```

Inmatning från tangentbord till program skrivs

```
Console.ReadLine() // string
```

typas ofta om till integer eller double t.ex.

```
x = int.Parse(Console.ReadLine());
```

behöver du matematiska funktioner har du tillgång till t.ex.

```
Math.Pow(,), Math.Sqrt(x),Math.Sin(x), Math.Cos(x),Math.Tan(x),Math.Exp(x)
```

Heltaldivision kapar decimalerna .
%-tecknet(modulo) används för att ge resten vid heltalsdivision.

Ibland måste vi kunna välja olika vägar beroende på om vissa villkor stämmer eller inte

```
if (uttryck)
{
        en eller flera satser
}
else if (annat uttryck)
{
       en eller flera satser }
else
       vad som  ska göras om både if och else if är fel
```

Jämförelse operatorer

Operator	Betyder	Exempel
<	Mindre än	if (a<20)
>	Större än	if (b>100)
==	Lika med	if (produkt==500)
!=	Skiljt från (inte lika med)	if (x!=28)
<=	Mindre än eller lika med	if (a<=2000)
>=	Större än eller lika med	if (b>=12)

Logiska operatorer

Operator	Betyder	Exempel
&&	och	if ((a>20) && (a<30))
||	eller	if ((a==0) || (b<0))
!	inte	if (!(a>0))

Vill vi upprepa några satser ett känt antal gånger använder vi oss av for loopen

```
for (initiering av startvärde; villkor; upp/ner-räkning ) {
      en eller flera satser;
}
t.ex.
for (int a=0;a<20;a++) {
      Console.WriteLine("nummer:" + a);
}
```

Vet vi inte när vi börjar loopen hur många varv som ska upprepas använder vi hellre while-loopen där vi kan ställa fler villkor

```
while ( uttryck som ska vara sant om det innanför klamrarna ska göras) {
     en eller flera satser;
 }
int x=0;
while x < 10) {
      Console.WriteLine("nummer:" + x);
      x++;

}
```

En annan variant av while-loopen: do-while

```
do
{
   en eller flera satser;
} while ( villkor som ska vara sant om det innanför klamrarna ska utföras)
```

behöver vi variabler som är indexerade använder vi oss av arrayer/listor/fält, deklareras t.ex.

```
int[ ] tal  =  new int[11];
double[ ] decimalserie  =  new double[50];
string[ ] namn = new string[5];
```

Slumptal

```
Random randomTal1 = new Random();
Console.WriteLine(randomTal1.Next());   //ger heltal mellan 0 och 2147483647
Console.WriteLine(randomTal1.Next(10); //ger heltal mellan 0 och 9
Console.WriteLine(randomTal1.Next(10,20); //ger heltal mellan 10 och 19
Console.WriteLine(randomTal1.NextDouble());  // ger decimaltal mellan 0 och 1, 0<=x<1
```

Metoder börjar med static när vi använder Console Application

Här är några exempel:

```
static int summa( int a, int b)
{
    return a+b;
}

static double max(double tal1, double tal2)
{
    if (tal1 > tal2) return tal1;
     else return tal2;
}

static void meny()
{
   Console.WriteLine("MENY");
}

static void kvadrera(ref int i)
{
    i = i * i;     // här kvadreras i, och alltså också tal i huvudprogrammet
}

static void skrivUt(int[] lista)
{
    for (int i = 0; i < lista.Length; i++)
        Console.Write(lista[i] + " ");
  }

 static int nFakultet(int n)
{
      if (n<=0) return 1;
      else return n*(nFakultet(n-1));
}
```