**Bokens Kapitel (vecka 1-2)**

**Kap 2: PRIMITIVA TYPER OCH VARIABLER**

* Primitiva typer och variabler
* Datatyper
* Variabler
* Value types – Referenstyper
* Literaler

**Kap 3: OPERANDER OCH UTTRCYK**

* Operander
* Konvertera datatyper
* Explicit/implicit
* Olika uttryck

**Kap 5: VILLKORLIGA UTTRYCK**

* Villkorliga uttryck
* if, else-if, switch
* Nestlade if-satser

**Kap 6: LOOPAR**

* For
* While
* Foreach
* Do-while

**Kap 7: ARRAYS**

**Kap 13: STRING OCH TEXTHANTERING**

**Olivias genomgång (vecka 1-2)**

1. **Vad är programmering**

* Hårdvara, mjukvara

1. **Typer, variabler**

* Bool

1. **Hur fungerar det INUTI datorn**

* Stack och Heap
* Garbage Collector (GC)

1. **Conditional statements**

* ”if” and ”if-else”
* Switch-case

1. **Loopar**

* for
* while
* do-while
* foreach

1. **Arrays**

* Singledimensionella, Tvådimensionella, multidimensionella
* Arrays som objekt
* Foreach loopa igenom arrays

1. **Metoder**
2. **Objekt –** Instanser
3. **Vad är programmering**

Programmering innebär att ge instruktioner till en dator som beskriver hur den ska utföra en given uppgift. För att få datorn att förstå **vad** vi vill att den ska göra tar vi hjälp av ett programmeringsspråk och skapar ett **program.**

* Hårdvara, mjukvara

**Program** kan vara skrivna på:

* ***Maskinkod*** (består av 1:or och 0:or (binärkod))
* ***Assembleringsspråk:*** Svårläst för människor, men enklare än maskinkod
* ***Högnivåspråk:*** C#, Java, Pyton, C, C++, Kobol

1. **Typer och variabler**

Bool, int, string

1. **Hur fungerar det INUTI datorn?**

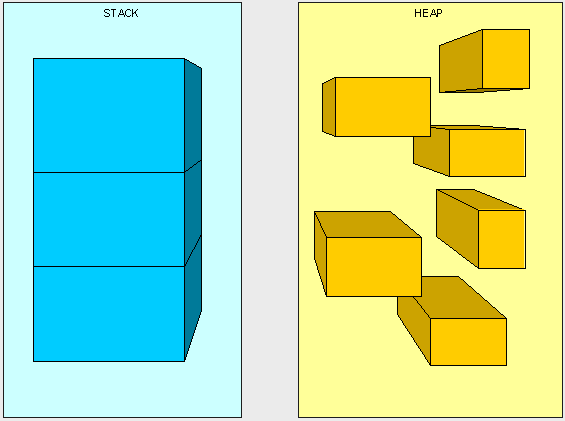
* Stack och Heap
* Garbage Collector (GC)

I .NET platformen behöver man inte förstå på en allt för komplicerad nivå hur minnet och papperskorgen fungerar. Dock är det bra att ha en grundläggande förståelse för detta för att skapa applikationer, optimera programmerandet samt för att förstå beteendet av variablerna vi skapar i programmen.

**Koden SPARAS på två ställen när vi stänger ner koden:**

* Stacken
* Heapen

**Stacken** tar hand om sig själv, **den tar hand om sitt minne själv**. När ”den översta skolådan” är färdiganvänd, kastas den ut.  
**Heapen** å andra sidan, måste ta hand om skräpet, (**G**arbage **C**ollector, **GC**). Garbage Collectorn är det som håller Heapen ren. (Ingen vill ha smutsig tvätt som ligger kvar!)



**VAD finns i Stacken och Heapen?  
  
- Value types  
- Referense typer  
- Pekare  
- Instruktioner**

**6. Arrays**

* En array kan vara **single-**, **multidimensionell** eller **Jagged**
* **Antalet dimensioner** och **längden** av varje dimension skapas när **instansen** skapas. Dessa värden kan inte ändras under arrayens livstid.
* **Elementens defaulta värden** på **numeriska arrays** är **noll**. Även om **referennselement** är satta till värdet noll.

**SINGLEDIMENSION ARRAYS**

**ARRAY DEKLARERING**  
**Int [ ]** array = **new int [5]  
En single-dimension array med fem heltal deklarerad**. Den här arrayen innehåller element från array[0] till array[4]. Operatorn **new** används för att skapa arrayen och **initiera arrayens element** till deras **defaulta** värde. I exemplet ovan är **alla elementen initierade** **till NOLL**

**String [ ]** stringArray = **new string [6]**  
Arrayen har sex stycken tomma platser.

**Deklarerar SAMTIDIGT som den initieras**  
**Int [ ]** array1 = **new int [ ]** {1, 3, 5, 7, 9};  
Då behövs inte ”rank specificeraren” inne i hakklamrarna new int [ ] - (5), eftersom det syns i initieringen efter.

**String [ ]** weekdays1 = **new string [ ]** {”Sun”, ”Mon”, ”Tus”, … , ”Sat”};  
Arrayelementenär initierat med veckodagarna

**Int [ ] array2 = new int [ ] {1, 3, 5, 7, 9};  
String [ ] weekdays2 = new string [ ] {”Sun”, ”Mon”, ”Tus”, … , ”Sat”};**

**Deklarerar UTAN initiering**  
**Int [ ]** array3;  
array3 = **new int [ ]** {1, 2, 3, 4, 5, 6, 7, 8, 9}; **// OK**  
 **// array3 = {1, 2, 3, 4, 5, 6, 7, 8, 9}; ERROR**Det går att deklarera en array variabel utan att initiera den på samma rad, men när man ska ge den dess värden måste man använda **new** operatorn.  **I C#, är arrays verkliga objekt,** (och inte bara addressable regions of contiguous memory som i C++ och C). [Array](https://docs.microsoft.com/en-us/dotnet/api/system.array) är den grundläggande typen av alla arraytyper. **Man kan använda arrayens ”properties/states/characteristics” (beskriver objektet i specifika situationer).** Arrayens **klassmedlemmar går också att använda.**

Exempel:   
int [ ] numbers = {1, 2, 3, 4, 5};  
int lengthOfNumbers = numbers.Lenght; **//** **ger längden på numrens array (5)**

**MULTIDIMENSIONEL ARRAYS**En mulitidimensionell array kan exempelvis vara tvådimensionell med 4 rader och 2 kolumner.

**TVÅDIMENSIONELL ARRAY [ , ]  
int [ ,]** twoDimArray = **new int [rows, columns]**;  
**int [ ,]** array2D = **new int [4, 2]**;

**Deklarerar tvådimensionell SAMTIDIGT som den initieras  
int [ , ]** array2D = **new int [ , ]** {{1, 2}, {3, 4}, {5, 6}, {7, 8}};  
 **//Samma array men som specificerar dimensionen  
int [ , ]** array2Da = **new int** **[2, 4]** {{1, 2}, {3, 4}, {5, 6}, {7, 8}};

**//Liknande array med stringelement  
string [ , ]** array2Db = **new string** **[3, 2]** {{”one”, ”two”}, {”three”, four”}, {”five”, ”six”}};

**TREDIMENSIONELL ARRAY [ , , ]  
int [ , , ]** threeDimArray = **new int [rows, columns, thirdDim];**  
**int [ , , ]** array3D = new int [4, 2, 3];

**Deklarerar tredimensionell SAMTIDIGT som den initieras**  
**int [ , , ]** array3D = **new int** **[ , , ]** {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}};  
  
**//Samma array men som specificerar dimensionen  
int [ , , ]** array3Da = **new int [2, 2, 3]** {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}};

// Getting the total count of elements or the length of a given dimension.

var allLength = array3D.Length;

var total = 1;

for (int i = 0; i < array3D.Rank; i++) {

total \*= array3D.GetLength(i);

}

System.Console.WriteLine("{0} equals {1}", allLength, total);

// Output:

// 1

// 2

// 3

// 4

// 7

// three

// 8

// 12

// 12 equals 12

**FOREACH LOOPA GENOM ARRAYS**

**ITERATING THROUGH AN ARRAY**int [ ] numbers = {4, 5, 6, 1, 2, 3, -2, -1, 0};  
**foreach**(**int** i **in** numbers)  
{

Console.Write({0}, i);

}

**// Output:** 4 5 6 1 2 3 -2 -1 0  
Ger oftast ut element på plats noll och upp till det sista elementet

**7. Metoder**

**Metoder kan delas in i olika grupper:**

* **Värdeanrop:** Metoder som **RETURNERAR** ett värde (och med parametrar)
* **Returnerar INTE** något värde
* Metoder med **DEFAULT VÄRDE**

**Metod som RETURNERAR:**Vi skickar information (ett eller flera värden) till metoden som i sin tur gör beräkningar MED DESSA VÄRDEN och sen RETURNERAR resultatet av dessa bearbetningar. **Vi skickar in ett eller flera ARGUMENT och metoden returnerar ett VÄRDE**.

**Ex:**  
int CountSum (int nr1, int nr2) **// Huvud**  
{  
 return (a+b); **// Kropp**  
}

**Metod som INTE RETURNERAR:**Denna metod returnerar INGET SVARSVÄRDE. Den skriver bara ut en text. **Inga argument skickas med när vi anropar metoden**, därav de tomma paranteserna och ordet **VOID.**  
 **Ex:**void Meny ()  
{  
 Console.Write(”MENY”);  
}

**Metod med DEFAULT VÄRDE:**Default-värden på parametrar, d.v.s. att i metoddeklarationen ange värden för parametrarna som ska antas om inget värde lämnas vid metodanropet: **Ex:**void SkrivUt (double tal, int counter = 1)  
{  
 ConsoleWriteLine(tal);  
}