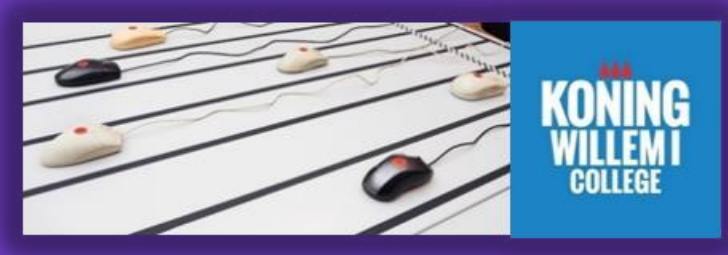
Basis standalone

Realiseren

hoofdstuk

2

Variabelen en logische structuren

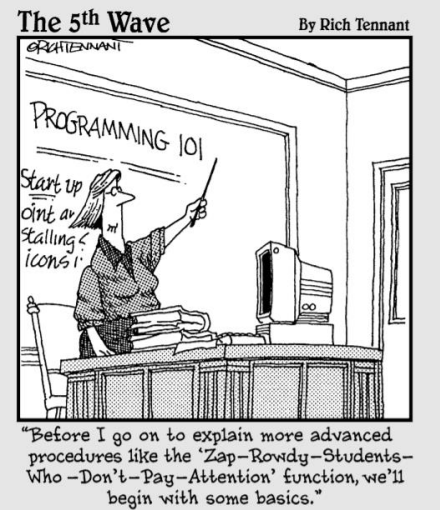


Algemene informatie

Onderwerp	Variabelen en Logische structuren
Leerdoel(en)	1. De student is in staat de datatypes in C# te gebruiken 2. De student is in staat een if/else statement in C# te gebruiken 3. De student is in staat om loop te gebruiken in C# 4. De student is in staat om arrays/lists te gebruiken in C#
Vereiste voorkennis	1. De student heeft kennis van de modules realiseren van thema 1 tm 4
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

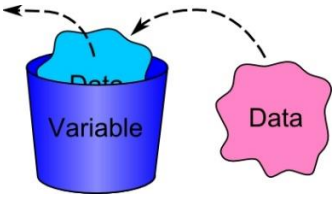
Introductie



Iedere programmeertaal heeft zijn manier om gegevens op te slaan in variabelen en verschillende soorten variabelen. Daarnaast heeft iedere programmeertaal de mogelijkheid om logica in te bouwen. In thema 2, 3 en 4 heb je geleerd hoe dit werkt in Javascript en PHP. In dit hoofdstuk herhalen we de leerstof over variabelen, datatypes en deze logische structuren en leggen we uit hoe je deze in C# toe past. Je zult zien en ontdekken dat in C#

Variabelen

In thema 1 van deze opleiding heb je al geleerd dat je een variabele gebruikt om waarden in op te slaan. Deze waarden kun je later weer gebruiken en/of aanpassen. Om de waarden makkelijk terug te vinden kun je een variabele een naam geven. Die naam is een verwijzing naar een locatie in het geheugen waar de waarde van de variabele wordt opgeslagen.



Elke programmeertaal maakt gebruik van variabelen. Variabelen kunnen waarden opslaan zoals naam, kleur, locatie. In C# maak je dat op de volgende manier aan:

```
string name = "Visual studio";
string color = "red";
string location = "Den Bosch";
```

Zoals je ziet lijkt dit heel erg op de manier waar je in Javascript geleerd hebt met variabelen te werken. Kijk nog eens terug in reader 4 van thema 1, zie jij de verschillen?

Variabelen maak je aan door ze te **declareren**.

```
string name = "Visual studio";
```

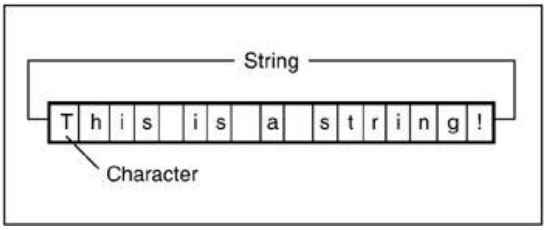
Daarna geef je ze een waarde door ze te **initialiseren**.

```
string name = "Visual studio";
```

Naamgeving variabelen
Net als in Javascript is het ook in C# belangrijk dat je zinvolle namen bedenkt voor variabelen. Het moet duidelijk zijn wat je er in opslaat. Een variabele met de naam ***variable1*** zegt niet zoveel. Een variabele met de naam ***age*** wel. Verder zijn er in C# verschillen in variabelen die public en variabelen die private zijn. Daar komen we later op terug, ook wat daarvan de gevolgen zijn voor de naamgeving.

Datatypen

Een datatype geeft aan wat voor soort data in de variabele wordt opgeslagen. In reader 4 van thema 1 (Basis statische website) heb je een aantal datatypes van Javascript geleerd. In C# heb je meer verschillende datatypes. Een aantal van deze datatypes zie je in onderstaande tabel:

Datatype	Omschrijving	Voorbeeld waarden
String	- Een reeks karakters. - Een string zet je altijd tussen aanhalingstekens. 	"Dit is een string" "Koning Willem I College" "123" "true" "Voorbeeld 5"
Int	- Een geheel getal, ook wel **integer** genoemd. - Een integer zet je niet tussen aanhalingstekens. Als je dat doet wordt het een string. Met strings kun je niet gaan rekenen.	1 6523 -1 -768
Double	- Een decimaal getal. - Decimale getallen bevatten een punt in plaats van een komma. - Een double zet je niet tussen aanhalingstekens. Als je dat doet wordt het een string. Met strings kun je niet gaan rekenen.	123.50 0.78996 -1.23 -9623.6543
Bool of Boolean	- Een variabele van het type bool of Boolean kent maar twee mogelijke waarden **true** of **false**. - Een boolean zet je niet tussen aanhalingstekens. Als je dat doet wordt het een string. Je slaat dan de tekst "true" of "false" op. Let op: allemaal kleine letters!	true false
DateTime	- Een variabele van het type DateTime bevat een datum EN tijd - Met een variabele van het type DateTime kun je wel 80 verschillende operaties uitvoeren, zoals alleen de maand laten zien, de dag van de week laten zien, dagen, uren, minuten erbij optellentwee datums van elkaar afhalen.	DateTime thisYear = new DateTime(2021, 1, 1); DateTime thisMoment = DateTime.Now; DateTime anHourFromNow = thisMoment.AddHours(1);

NULL-waardes

Wanneer een variabele geen waarde heeft, krijg je vaak een default waarde te zien. Zo is een lege string bijvoorbeeld aangeduid als volgt: “”. Dit noemen we een lege string (de string is correct opgebouwd, echter is de waarde nog leeg). Soms kom je echter zogeheten **NULL**-waardes (uitgesproken als NIL-waarde) tegen. NULL is een soort datatype waarin aangegeven wordt dat een waarde absoluut mist. Je zult merken dat een We gaan nu niet dieper in op deze NULL, maar laten onderstaande plaatje de uitleg doen:

De programmeertaal C# is veel strikter dan de taal Javascript. Een variabele is van een bepaald type en dat kan niet veranderen. In Javascript kon je in een variabele eerst een **string** plaatsen en vervolgen een **number**. Dat accepteert C# niet. Hoe kunnen we dan bijvoorbeeld een getal maken van een ingevoerde string? Daarvoor moet we de string omzetten via het door .NET ingebouwde klasse **Convert.** Een voorbeeld van het omzetten naar een *integer* zie je hieronder:

```
// declaratie van variabele input, datatype string
string input;
// Plaats de ingevoerde tekst in de variabele input
input = Console.ReadLine();
// declaratie van variabele amount, datatype int
int amount;
// zet de tekst in variabele input om naar een getal en plaats
// dat in de variabele amount
amount = Convert.ToInt32(input);
```

Via de klasse **Convert** kun je variabelen omzetten naar allerlei andere datatypes (int, double, boolean, DateTime en nog vele meer).

Standaard eigenschappen voor strings

Net als in bv. Javascript heeft een string datatype in C# ook allerlei eigenschappen. Bijvoorbeeld het aantal letters van de string (de lengte).

.Length
Met de standaard eigenschap .length kun je het aantal karakters van een string ophalen.

```
string schoolName = "KW1C";
int schoolNameLength = schoolName.Length;
```

Standaard methods voor strings

In Javascript hebt je al geleerd wat functies zijn in die programmeertaal (zie anders reader 4 van thema 1). In C# kun je ook gebruik maken van functies, al noem je die in C# geen functie (function) maar method. Je herkent een method in C# aan de ()-tekens.

Er zijn ook in C# een heel aantal functies die alleen bedoelt zijn om strings aan te passen. Denk daarbij aan omzetten naar hoofdletters of kleine letters, specifieke letters opzoeken of vervangen etc. Hieronder volgen een paar voorbeelden van functies waarmee je strings kunt aanpassen.

.toUpper ()
Met de standaard functie .toUpper() kun je een string omzetten naar hoofdletters.

```
string schoolName = "KW1C";
string schoolNameUpper = schoolName.ToUpper();
```

.toLower ()
Met de standaard functie .toLower() kun je een string omzetten naar kleine letters.

```
string schoolName = "KW1C";
string schoolNameLower = schoolName.ToLower();
```

.Substring()
Met de standaard functie .substring() kun je een deel van een string ophalen.

```
string schoolName = "KW1C";
string schoolNamePart = schoolName.Substring(0,3);
```

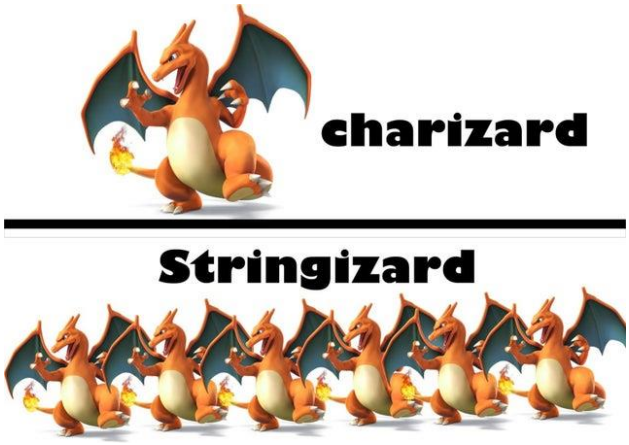
In de string schoolNamePart komt hier de waarde KW1 te staan.

Je ziet dat er tussen de haakjes waarden zijn opgegeven. Dat noemen we, net als in javascript, parameters. Je geeft met deze waarden aan bij welke letter je wilt starten en hoeveel letters je wilt meenemen. De computer telt hierbij vanaf 0.

0	1	2	3
K	W	1	C

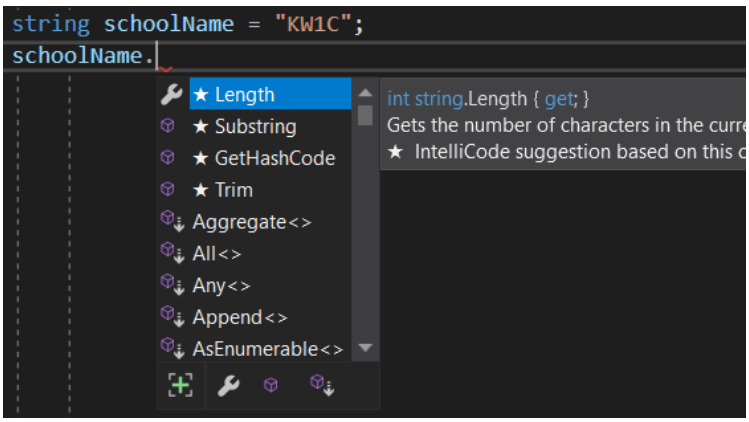
Een String is een reeks van Chars.

Je kent het datatype genaamd **Char** waarschijnlijk nog wel uit de SQL-stof. Een Char staat voor één karakter. Een String is dus slechts een reeks (Array) aan karakters.



Overige string methods

Er zijn veel meer methodes die je voor een string kunt uitvoeren. Probeer maar eens een string variabele te maken, type achter de naam van de variabele een punt en visual studio laat je de mogelijkheden zien die je kunt kiezen!



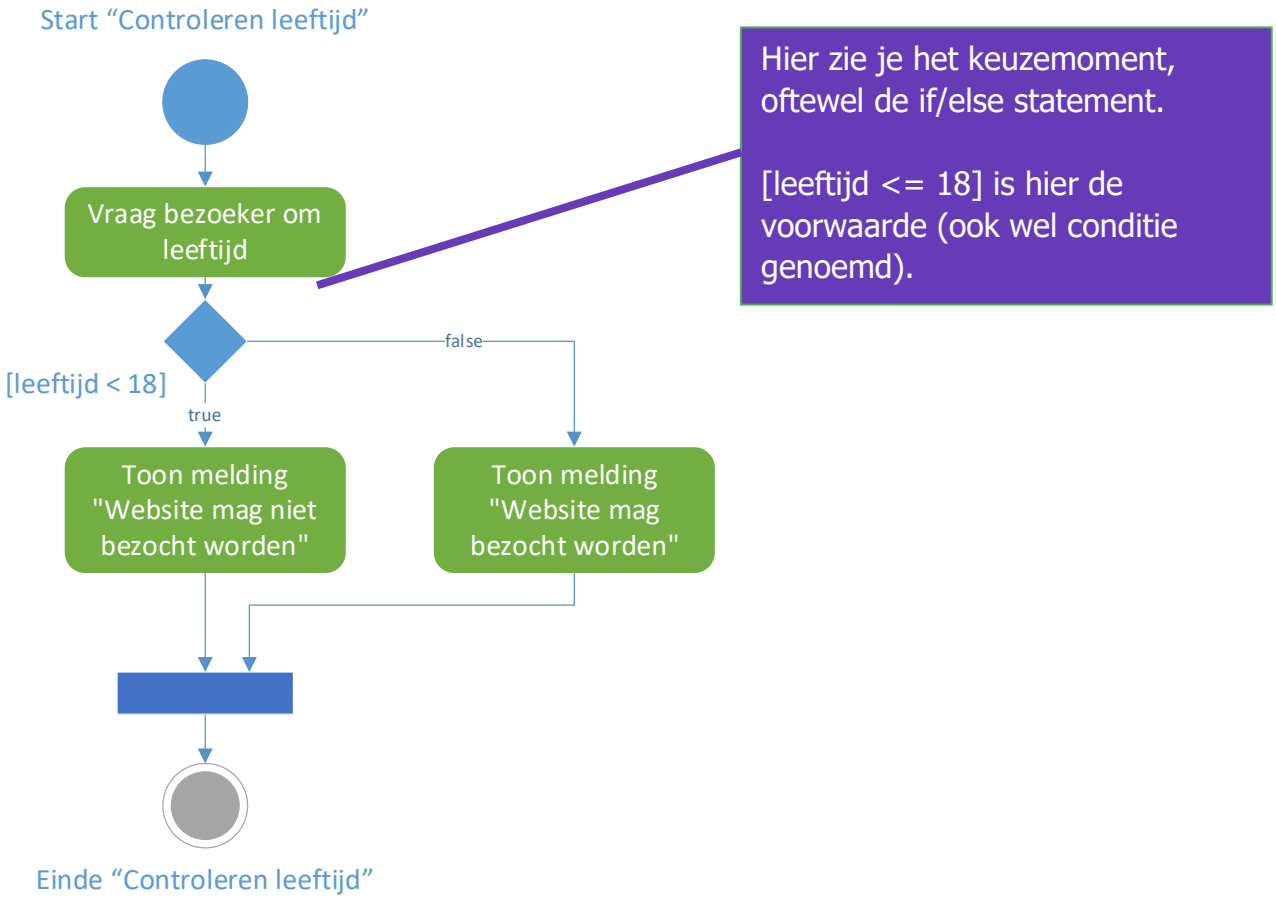
Vergelijkingsoperatoren

Vergelijkingsoperatoren kunnen gebruikt worden om waarden met elkaar te vergelijken. De volgende operatoren kun je gebruiken in C#.

Vergelijkingsoperator	Betekenis	Voorbeeld	Resultaat
==	Gelijk aan	1 == 1 "1" == 1	True True
!=	Niet gelijk aan	1 != 2 1 != 1	True False
>	Groter dan	5 > 1	True
<	Kleiner dan	1 < 3	True
>=	Gelijk aan, of groter dan	5 >= 1 5 >= 5	True True
<=	Gelijk aan, of kleiner dan	1 <= 3 3 <= 3	True True

If/else statement

In reader 4 van thema 1 heb je geleerd hoe je een keuze kan programmeren. Dat doe je met een if/else statement. Op de volgende bladzijde zie je een eenvoudig activiteitendiagram met een keuzemoment.



Bovenstaande activiteitendiagram kan op de volgende manier worden omgezet naar C# code.

```
// Vraag bezoeker om leeftijd
Console.WriteLine("Hoe oud ben je ?");
int age = Convert.ToInt32(Console.ReadLine());
// Als age onder de 18 is
if (age<18)
{
    // Dan wordt deze code uitgevoerd
    Console.WriteLine("Applicatie mag niet gestart worden. ");
}
else
{
    // Anders wordt deze code uitgevoerd
    Console.WriteLine("Applicatie mag gestart worden. ");
}
```

Hopelijk zie je de overeenkomsten in dit if/else statement en het if/else statement wat je al een aantal keer in Javascript hebt gemaakt.

Arrays

Arrays zijn in iedere programmeertaal dé standaard om lijsten met gegevens (zoals leerlingen, boodschappen, etc) op te slaan. Een Array kan een lijst van een bepaald type data opslaan (String, Integer, maar ook Objecten van eigengemaakte classes). In C# is er een standaard datatype Array[], die we eerst gaan bestuderen. We zullen hier wat onhandigheden in tegenkomen.

```
string[] weekDays = new string[7];
weekDays[0] = "Zondag";
weekDays[1] = "Maandag";
weekDays[2] = "Dinsdag";
weekDays[3] = "Woensdag";
weekDays[4] = "Donderdag";
weekDays[5] = "Vrijdag";
weekDays[6] = "Zaterdag";
```

In de eerste regel wordt de array *weekDays* gemaakt. Er kunnen maximaal 7 waarden in deze array worden opgeslagen. Daarna worden alle rijen gevuld met de dagen van de week.

Zoals je weet, start een Array index altijd bij 0.

LET OP

Het is niet mogelijk om in C# een Array uit te breiden met extra gegevens (indexnummers). Hiervoor moet je gebruik maken van andere datatypes zoals een List.

Je hebt hierboven gezien hoe je een array kunt aanmaken en vullen met gegevens. Hieronder worden nog eens kort alle onderdelen herhaald en uitgelegd.

De onderstaande instructie declareert een array van cijfers (we gebruiken even een Nederlandstalige variabelenaam *rapportcijfers* voor de duidelijkheid):

```
int[] rapportcijfers;
```

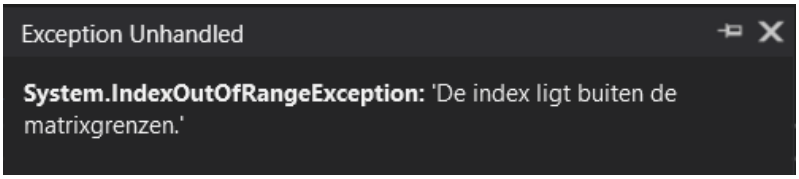
Het aantal variabelen, in dit geval getallen, dat je in de array kan plaatsen is nog niet bekend. Je kunt dus nog niets in de array plaatsen. Met de volgende code los je dat probleem op:

```
rapportcijfers = new int[3];
```

Nu kan je drie cijfers in de array plaatsen. Je moet er rekening mee houden dat de index van een array altijd met 0 start. Je voegt nu als volgt drie cijfers toe:

```
rapportcijfers[0] = 5;
rapportcijfers[1] = 7;
rapportcijfers[2] = 6;
```

Als je probeert een vierde cijfer in de array te plaatsen gaat het mis. Je krijgt echter pas tijdens het debuggen een foutmelding hierover:



De error die je krijgt, als je een item probeert toe te voegen die buiten de maximale Index ligt.

List

Je hebt gezien dat een **Array** datatype altijd een vast aantal waardes moet hebben. Als je bijvoorbeeld een Array declareert en initialiseert op de volgende wijze, kun je nooit meer dan 3 items aan de Array toevoegen. Dit is erg vervelend en onhandig.

```
string[] shoppingItems = new string[3];
```

Hierboven heb je gezien dat als je aan bovenstaande Array een item op positie 4 probeert toe te voegen krijg je Out of Range error. Je hebt de zogeheten **Upper Bound** (Maximale index waarde) van de Array.

Daarnaast zijn er nog andere onhandige zaken rondom het standaard Array datatype. Om deze problemen op te lossen, is er een verbetering van het Array datatype ontwikkeld. Deze verbetering heet **List** en is een klasse die we gebruiken om het werken met Arrays een stuk makkelijker te maken.

Aanmaken List

Een nieuwe **List** met strings aanmaken om een lijst van strings te kunnen beheren, doe je op de volgende manier:

```
List<string> shoppingItems = new List<string>();
```

Je declareert EN initialiseert hier een List, waarvan de items van het datatype *string* zijn. Binnen de haken <> geef je het datatype op. Dit kan dus ieder datatype zijn! Hierop komen we in andere hoofdstukken nog op terug.

Daarna kun je items toevoegen aan de **List** *shoppingItems* via de methode **.Add()** op het List object. Dat doe je via onderstaande manier:

```
shoppingItems.Add("Melk");
shoppingItems.Add("Handgel");
shoppingItems.Add("Hagelslag");
```

Je kan dus zoveel items toevoegen als je wilt en hoeft dus niet (zoals bij een Array) vooraf op te geven uit hoeveel items je lijst zal bestaan.

Uitlezen List
Je kunt (net als bij een reguliere Array) één item per keer uitlezen door via de blokhaken [] een index op te geven. De index start uiteraard altijd bij 0. Zie voorbeeld hieronder:

```
string item1 = shoppingItems[0];
string item2 = shoppingItems[1];
string item3 = shoppingItems[2];
```

List items aanpassen
Het is mogelijk om één specifiek item aan te passen. Ook dit kun je doen door weer de blokhaken [] te gebruiken en hierbij de index op te geven.

```
shoppingItems[2] = "Toiletpapier";
```

Loops

Met een loop kun je een stuk code vaker laten uitvoeren met eventueel veranderende waarden. Er zijn, net als in Javascript en PHP, verschillende soorten loops in C# zoals de foreach, while en de for. Een for-loop gebruik je als je van te voren al weet hoe vaak je de code wilt laten herhalen. Een while loop gebruik je op het moment dat je een code wilt laten uitvoeren zolang er een aan bepaalde voorwaarde wordt voldaan, maar je niet weet wanneer dat stopt. Bijvoorbeeld als een gebruiker een timer moet kunnen laten stoppen. Je weet dan niet wanneer de gebruiker op een stop knop drukt.

For
In onderstaand voorbeeld wordt een for-loop gebruikt om 100x dezelfde zin in een paragraaf te zetten.

```
string punishmentText = "";

for (var counter = 1; counter <= 100; counter = counter + 1)
{
    punishmentText = punishmentText + $"Ik moet deze zin 100x overschrijven! \n";
}

Console.WriteLine(punishmentText);
```

Zoals je hier ziet lijkt deze code erg op de for-loop zoals je dit bij bijvoorbeeld Javascript hebt geleerd. Weet je niet zo goed meer hoe een for loop werkt, verwijzen we je naar reader 4 van thema 2.

Net als in javascript kan ook in C# de counter van een for-loop gebruikt worden binnen het codeblok. Zo kun je bijvoorbeeld ook van 1 tot 100 tellen en dat laten zien.

```
string countText = "";

for (var counter = 1; counter <= 100; counter = counter + 1)
{
    countText = countText + counter.ToString();
}

Console.WriteLine(countText);
```

De gebruikte for-loop kan nog wat worden ingekort, maar dat maakt de for-loop wel minder goed te begrijpen. Kies dus zelf wat je prettig vind, makkelijker te lezen code, of kortere code.

```
string countText = "";

for (var i = 1; i <= 100; i++)
{
    countText = countText + i.ToString();
}

Console.WriteLine(countText);
```

Het maakt voor de werking van de code natuurlijk niets uit hoe je de variabele noemt. In een for-loop wordt meestal de variabele i gebruikt voor een counter. De i staat voor iteration (herhaling). In plaats van i = i + 1, kun je ook zeggen i++. Dit doet precies hetzelfde, namelijk 1 bij de huidige waarde van de variabele i optellen.

While

Net zoals in Javascript kun je ook in C# een while loop maken. Deze lijkt op de for loop, maar heeft, net als in Javascript, een paar kleine verschillen.

For loop:

```
string countText = "";

for (var i = 1; i <= 100; i++)
{
    countText = countText + i.ToString();
}

Console.WriteLine(countText);
```

While loop:

```
string countText = "";
int i = 1;

while (i <= 100)
{
    countText = countText + i.ToString();
    i++;
}

Console.WriteLine(countText);
```

Tussen de haakjes achter de while staat nu alleen de conditie. Het aanmaken van de teller doe je voordat je de loop maakt. Het ophogen van de teller doe je in het codeblok. Weet je niet zo goed meer hoe een while loop werkt, verwijzen we je naar reader 4 van thema 2.

Array/List en loops

Loops worden ook vaak gebruikt om waarden uit een array/list te doorlopen. Daarbij wordt een teller van een loop gebruikt als waarde van een index.

```
string resultText = "";
List<string> subjects = new List<string> { "PO", "REA", "TV", "COM", "PRA" };

for (int i = 0; i < subjects.Count; i++)
{
    resultText = resultText + subjects[i] + $"\n";
}

Console.WriteLine(resultText);
```

Er wordt eerst een list aangemaakt met daarin 5 items.

Index	Item
0	PO
1	REA
2	TV
3	COM
4	PRA

In de regel van het maken van de for-loop wordt een counter aangemaakt (variabele i) die op 0 gezet wordt. Dit wordt op 0 gezet, zodat dit in de code van de for-loop ook gebruikt kan worden om een index van de array mee aan te geven. De voorwaarde die wordt ingesteld is dat de counter kleiner moet zijn aan de lengte van de array (5). Omdat na het uitvoeren van de code de teller met 1 wordt opgehoogd wordt de code in de for-loop 5x uitgevoerd:

De teller start bij 0, i heeft dus de waarde 0. In de code wordt subjects[0] toegevoegd aan de variabele resultText, gevolgd door een teken wat de volgende tekst op een nieuwe regel zet.

Ook deze loop kan herschreven worden naar een while loop:

```
string resultText = "";
List<string> subjects = new List<string> { "PO", "REA", "TV", "COM", "PRA" };
int i = 0;

while (i < subjects.Count)
{
    resultText = resultText + subjects[i] + $"\n";
    i++;
}

Console.WriteLine(resultText);
```

Omdat je echter weet hoe vaak de loop uitgevoerd gaat worden is het dus ook hier netter om een for-loop te gebruiken.

Items benaderen via een foreach loop

Je kent al *while* en *for* loops uit andere programmeertalen zoals PHP en Javascript.

Maar wist je dat dit veel makkelijker gedaan kan worden via een **foreach**? Een **foreach** kan namelijk volledig automatisch door een **Array** / **List** heen loopen. Zie het voorbeeld hieronder:

```
string resultText = "";
List<string> subjects = new List<string> { "PO", "REA", "TV", "COM", "PRA" };

foreach (string subject in subjects)
{
    resultText = resultText + subject + $"\n";
}

Console.WriteLine(resultText);
```

De **foreach** gaat letterlijk "in" de List shoppingItems kijken en gaat per iteratie de variabele genaamd *item* vullen met de gegevens van dat item. Dit proces herhaalt zichzelf totdat alle items uit de List zijn gedaan.

Maak nu **Oefening 2.1**, om te oefenen met datatypes, loops, if/else etc.

Methods

Zoals binnen bijna iedere programmeertaal is het mogelijk om code te groeperen en te "parkeren", om deze daarna op een later moment te laten uitvoeren. Dit is handig, omdat je code slechts éénmaal hoeft te schrijven om deze code daarna meerdere malen uit te laten voeren.

Hiervoor gebruiken we **Methods**. **Methods** ken je waarschijnlijk al uit Javascript en PHP. Binnen deze scripttalen, werden ze **functions** genoemd. Lees hier meer over functions binnen PHP om je weer even op te frissen.

Het meest simpele voorbeeld van een method zie je hieronder. De method genaamd *GreetUser()* toont een melding aan de gebruiker. Je ziet de 2 *Console.WriteLine* statements die gegroepeerd zijn onder de naam *GreetUser()*.

```
public void GreetUser()
{
    Console.WriteLine("Welkom in onze omgeving.");
    Console.WriteLine("Hopelijk heeft u een fijn verblijf");
}
```

Belangrijk om te weten is, dat de code op dit moment nog niet uitgevoerd gaat worden. De code (de 2 regels aan *WriteLine*) gaat pas uitgevoerd worden, op het moment dat de methode *GreetUser* **aangeroepen** wordt.

```
// Voer de code binnen de methode GreetUser uit (aanroepen)
GreetUser();
```

```
Microsoft Visual Studio Debug Console
Welkom in onze omgeving.
Hopelijk heeft u een fijn verblijf
```

In het voorbeeld hierboven zie je dat de methode *GreetUser()* meerdere malen uitgevoerd kan worden. Je ziet dus, dat de methode slechts éénmaal aangemaakt hoeft te worden maar zo vaak als nodig is aangeroepen kan worden.

```
// Voer de code binnen de methode GreetUser uit (aanroepen)
GreetUser();
GreetUser();
GreetUser();
```

Uitvoer:

```
Microsoft Visual Studio Debug Console
Welkom in onze omgeving.
Hopelijk heeft u een fijn verblijf
Welkom in onze omgeving.
Hopelijk heeft u een fijn verblijf
Welkom in onze omgeving.
Hopelijk heeft u een fijn verblijf
```

Methods met parameters

Het is mogelijk om variabelen mee te sturen naar een methode (als input van de methode). Dit is nodig, omdat de code binnen een methode altijd hetzelfde doet, en door gebruik te maken van **Parameters** kan de Method tot andere berekeningen of uitkomst komen.

Hieronder zie je de methode *GreetUserWithName*(string). Deze methode ontvangt 1 parameter genaamd *name.*

```
public void GreetUser(string name)
{
    Console.WriteLine("Goedemorgen, " + name);
}
```

Hieronder zie je een voorbeeld van hoe je de Method GreetUserWithName(string) aanroept, inclusief een parameter.

```
// GreetUserWithName met parameter
GreetUserWithName("Ron");
```

Overigens is het ook mogelijk om een parameter mee te geven via een *variabele*. Zie het voorbeeld hieronder:

```
// GreetUserWithName met parameter (verpakt in een variabele)
string secondName = "Ron";
GreetUserWithName(secondName);
```

Dus de 2 aanroepen van hierboven geeft als resultaat:

```
Goedemorgen, Ron
Goedemorgen, Ron
```

Je kunt ook meerdere parameters meegeven (*string* user1*, string* user2 en *int* loginAmount):

```
public void ShowUserInfo(string user1, string user2, int loginAmount)
{
    Console.WriteLine("Welkom gebruiker 1: " + user1);
    Console.WriteLine("Welkom gebruiker 2: " + user2);
    Console.WriteLine("Aantal keer ingelogt: " + loginAmount);
}
```

Aanroepen van bovenstaande methode:

```
ShowUserInfo("Abu", "Remco", 15);
```

Return waardes

Een Methode voert code uit, en kan daarna bepaalde variabele als output hebben (zie schema hieronder). Zo'n output variabele noemen we een **return.** Eén specifieke methode kan maximaal één variabele als return waarde hebben.

Hieronder zie je een voorbeeld:

```
public string AskName()
{

    string output = "";

    Console.WriteLine("Geef uw naam op:");
    output = Console.ReadLine();

    // Return
    return output;
}
```

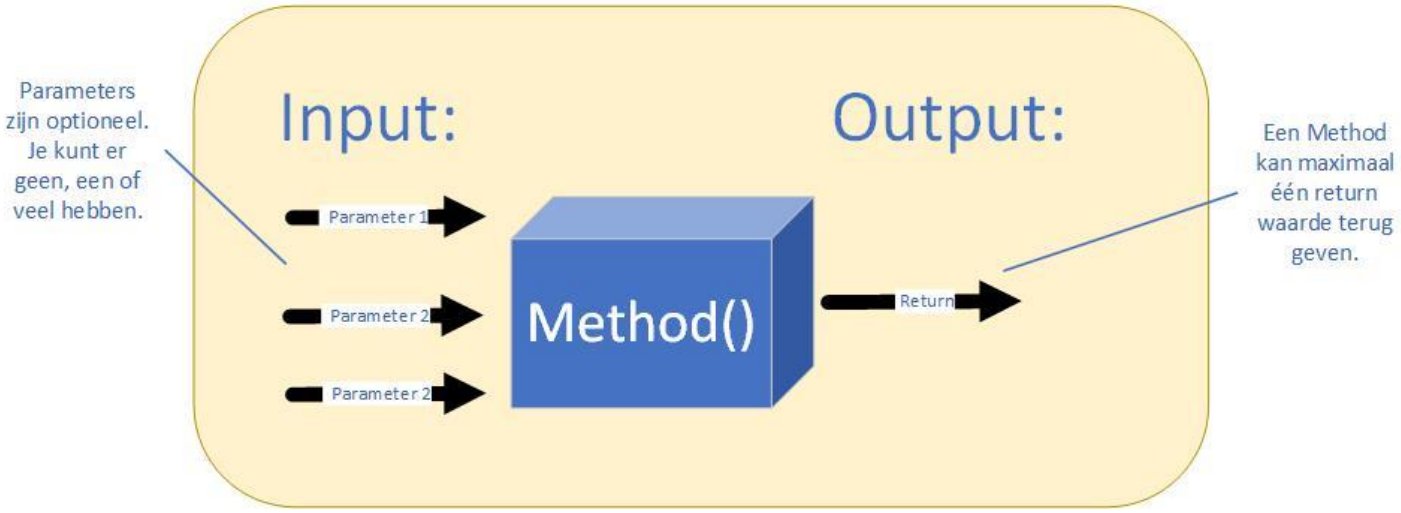
Hierboven zie je, dat je een Method aanmaakt genaamd **AskName()** die altijd een *string* dient te returnen. Doe je dat niet, dan krijg je onderstaande error:

```
string AskName()

                string Program.AskName()
                S0161: 'Program.AskName()': not all code paths return a value
put = "";

iteLine("Geef uw naam op:");
```

En hieronder zie je een voorbeeld van hoe je de methode **ReadLine()** ook een return gebruikt om jou een string terug te geven. Eigenlijk heb je dus als heel vaak de return gebruikt, zonder dat je het door hebt!

```
string userInput = Console.ReadLine();
```

Schematische opzet van een Method



Maak nu **Oefening 2.2**, om te oefenen Methods