

Hand-out

# **Programmeren**

04. Functies & Methodes

## 1. Inleiding

Een functie of een methode is een groep of insluiting voor meerdere regels code. Een functie heeft een naam en via deze naam kun je de functie uitvoeren in je programma.

In de functie kun alle code neer zetten die je wil gebruiken, er zit geen limiet aan maar er zijn wel bepaalde regels die je moet volgen.

Hier volgt een voorbeeld van een simpele statische constructor methode :

```
1. static void Main(string[] args)
2. {
3.     Console.WriteLine("Voer een waarde in :");
4.     var strResult = Console.ReadLine();
5.     Console.WriteLine("Je hebt net : " + strResult + " ingevoerd!");
6.     Console.ReadLine();
7. }
```

In dit voorbeeld zie je een static methode genaamd Main zonder return type, void (leeg). We weten inmiddels dat dit een constructor word genoemd. Void wil zeggen dat we geen waarde terug geven op het moment dat de functie van buitenaf wordt aangeroepen of uitgevoerd.

## 2. Scope

Een functie of methode heeft een scope, dat betekent, de bereikbaarheid van een functie kan variëren. De scope geldt voor functies het zelfde als voor variabelen.

- **Private** : Is alleen bereikbaar binnen de class waarin de functie is gedefinieerd.
- **Protected** : Is bereikbaar binnen de class waarin de functie is bereikt en alle classes die daar van overerven.
- **Public**: Is in iedere class bereikbaar.

De rede dat we iets private of protected willen maken is dat we niet willen dat derde partijen bij onze code kunnen en deze eventueel kunnen veranderen. We maken dan de meeste functies private of protected en delen alleen functie die we public maken die exact de data terug geeft die we willen delen met de rest van de wereld.

Een functie kan ook worden gedefinieerd als **static**, dit houdt in dat er geen instantie van de class gemaakt hoeft te worden (meer daar over later) om gebruik te maken van de functie, maar de functie kan direct via de classname worden uitgevoerd. Let wel op wanneer je dit doet moet je ook de variabelen die je global definieert static maken, anders mag je ze in een static methode niet gebruiken.

De Scope geldt ook voor methodes die niet static zijn, en voor variabelen die gedefinieerd worden in de class (global) en dus niet methode specifiek zijn. Variabele die binnen een methode worden aangemaakt en toegewezen kunnen ook alleen in die methode worden gebruikt, tenzij je deze als parameter door gaat sturen naar andere methodes, meer hier over in het volgende stuk.

Hier volgt een voorbeeld van een statische methode :

```
1. class Program
2. {
3.     public static int intResult;
4.
5.     static void Main(string[] args)
6.     {
7.         Console.WriteLine("Dit is een uitleg over functies");
8.         Program.intResult = Program.ShowInfo(Console.ReadLine(), 10);
9.         Console.WriteLine("De uitkomst is : " + intResult);
10.        Console.ReadLine();
11.    }
12.}
```

Zoals je ziet is er een variabele aangemaakt boven de functie, dit is een member de class geworden en is dus zogenaamd "global" gedefinieerd. Dit wil zeggen dat het binnen de hele class gebruikt kan worden en dus ook in iedere methode binnen die class. Omdat we de variabele willen gebruiken binnen een static functie moeten we ook de variabele static definiëren, dit is nu eenmaal de regel en heeft te maken met iets wat we een instantie van een class noemen, later meer daar over.

### 3. Parameters

In het voorbeeld hier boven zie je op regel 5 staan "string[] args", dit is een parameter van het type string array. Parameters zijn variabele die je aan een methode mee kunt geven. Als je een methode hebt gemaakt in je programma kun je er voor kiezen bepaalde data nodig te hebben om de methode uit te voeren, deze data wordt gedefinieerd als een of meerdere parameters.

Technisch gezien worden de variabelen die in de methode definitie staan argumenten genoemd en waar de methode word aangeroepen worden Parameters mee gegeven, voor het gemak zeggen we over beide gewoon dat het parameters zijn.

Een parameter is een variabele en bij het aanmaken van nieuwe variabelen hebben we geleerd dat we altijd een type moeten specificeren. Dit geldt ook voor een parameter, een parameter wordt namelijk ook nieuw aangemaakt op de plek waar je de functie of methode hebt gemaakt.

Let op met het gebruiken van parameters, iedere keer als je een parameter door geeft van de ene functie naar de andere maak je in principe een kopie van die variabele, dit houdt in dat wanneer de originele variabele veranderd het kopie er van niet veranderd. Als je dit probleem wil voorkomen kun je een variabele doorsturen als referentie, dit noemen we "byref". Hier voor gebruiken we het keyword "ref" of "out", later meer daar over.

Parameters zijn altijd verplicht als deze eenmaal zijn gedefinieerd maar, een parameter hoeft niet verplicht te zijn als we deze een standaard waarde mee geven, Hier volgt een voorbeeld :

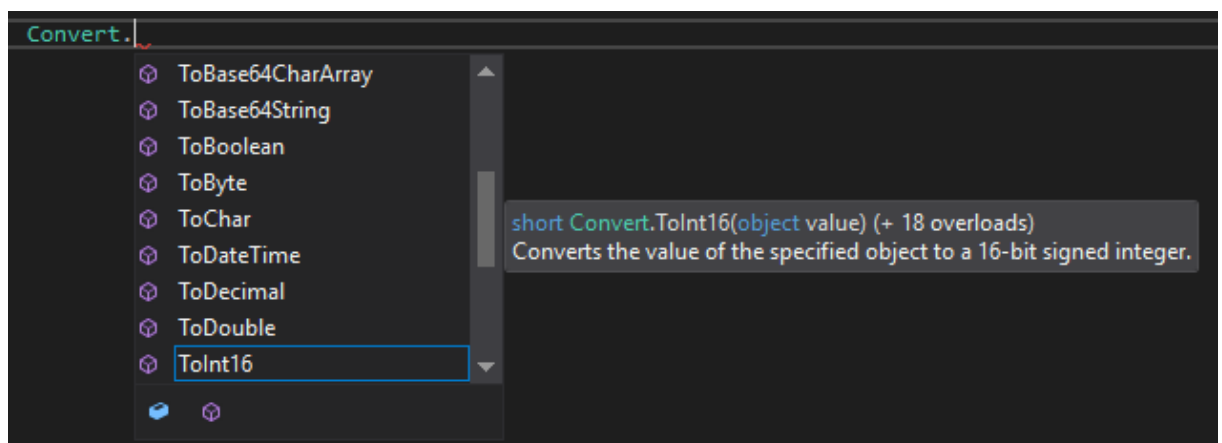
```
1. static void Main(string[] args)
2. {
3.     // resultaat : 16 / 2 = 8
4.     double dblResult = this.Divide();
5.     //resultaat : 100 / 2 = 50
6.     double dblResult = this.Divide(100);
7.     //resultaat : 300 / 100 = 3
8. }
9.
10. public double Divide(double dblA = 16, double dblB = 2)
11. {
12.     return(dblA / dblB);
13. }
```

Op MSDN kun je meer informatie vinden over parameters :

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/passing-parameters>

#### 4. Intellisense

Intellisense is een techniek die in Visual Studio zit en tegenwoordig in de meeste IDE's. Het is een hulpstuk voor programmeurs om al van te voren inzicht te krijgen in de mogelijkheden die een class of een bibliotheek te bieden heeft. Plus het geeft inzicht in het gebruik er van. Om intellisense te gebruiken hoef je niet veel te doen, het is standaard beschikbaar. Om intellisense op je scherm te krijgen gebruik je ctrl + spatie. Of je kunt intellisense openen door een classnaam in te typen en vervolgens een . te zetten, zoals hier onder in het voorbeeld :

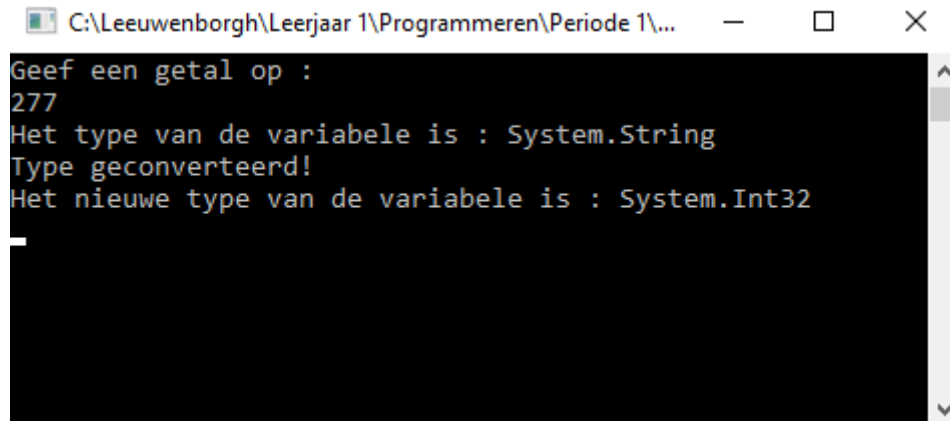


Meer over intellisense op MSDN :

<https://msdn.microsoft.com/en-us/library/hcw1s69b.aspx>

## 5. Demo

In dit stuk gaan we klassikaal een demo maken. Het is de bedoeling dat je mee doet en zo goed te weten komt hoe we variabelen definiëren en er data aan toe kennen , daarna ga je zelf aan de slag dus let goed op!



```
C:\Leeuwenborgh\Leerjaar 1\Programmeren\Periode 1\...
Geef een getal op :
277
Het type van de variabele is : System.String
Type geconverteerd!
Het nieuwe type van de variabele is : System.Int32
```

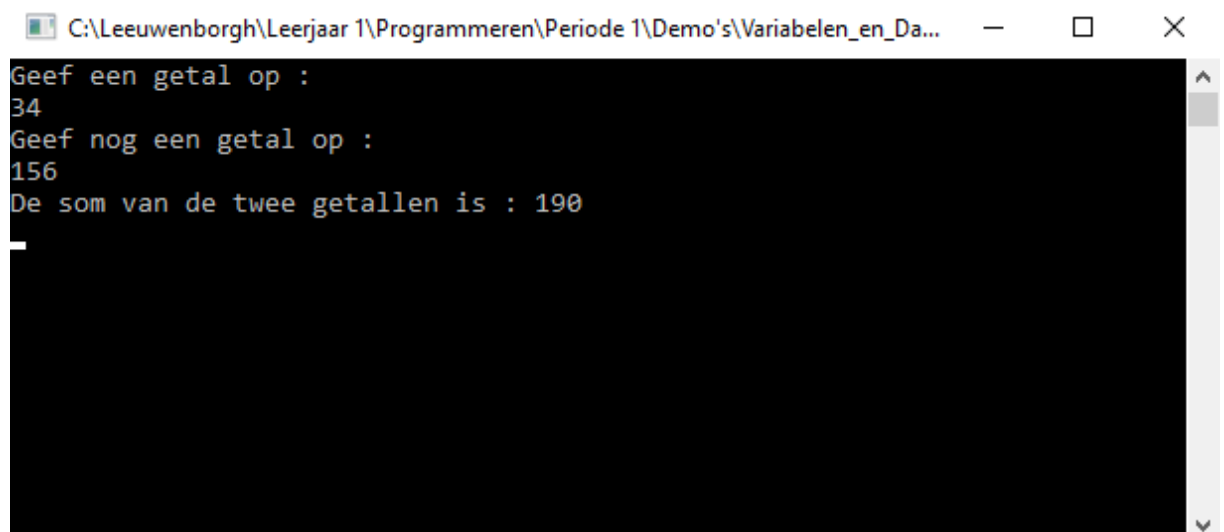
De uitwerking van de demo kun je downloaden op [GitHub](#).

## 6. Opdracht

Maak een programma dat vraagt twee keer vraagt om een getal in te voeren, sla deze getallen beide op in een eigen variabele. Laat vervolgens de som van de twee getallen op het scherm zien door deze bij elkaar op te tellen als volgt :

"De som van de getallen is <som>"

De woorden tussen de haakjes moeten natuurlijk vervangen worden met de som van de twee getallen. Als je klaar bent met deze opdracht ga je je werk samen met de docent bekijken. veel succes!



```
C:\Leeuwenborgh\Leerjaar 1\Programmeren\Periode 1\Demo's\Variabelen_en_Da...
Geef een getal op :
34
Geef nog een getal op :
156
De som van de twee getallen is : 190
```

## 7. Uitwerking

Demo code :

```
1. Console.WriteLine("Geef een getal op : ");
2. string strNumber = Console.ReadLine();
3. Console.WriteLine("Het type van de variabele is : " +
    strNumber.GetType().ToString());
4. int intNumber = Convert.ToInt16(strNumber);
5. Console.WriteLine("Type geconverteerd!");
6. Console.WriteLine("Het nieuwe type van de variabele is : " +
    intNumber.GetType().ToString());
7. Console.ReadLine();
```

De uitwerking van de opdracht volgt de volgende les!