C# Delegates, events

Versie 2020-2021

HO
GENT

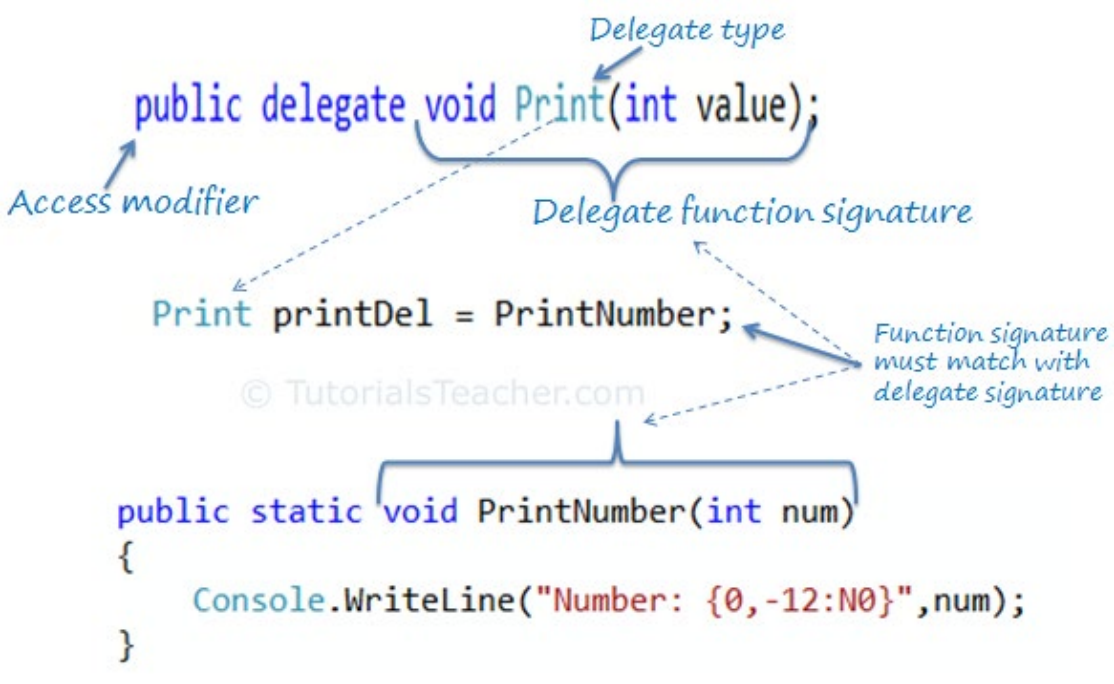
Delegate

- Een functie kan een of meer parameters van verschillend type hebben, maar wat als je een functie zelf als parameter wenst mee te geven?
- Een delegate refereert aan een functie, bevat een referentie naar een functie.
- Een delegate is afgeleid van class *System.Delegate*.
- Syntax:
  - <access modifier> delegate <return type> <delegate_name>(<parameters>)
- Voorbeeld:
  - Public delegate void Print(int value);

- Een delegate is een referentietype dat naar een methode refereert in plaats van naar een object. Wanneer we een delegate instantieren, dan associeren we de instantie van een methode die dezelfde signatuur heeft. De methode kan dan worden uitgevoerd door middel van de instantie. Delegates worden gebruikt om methodes door te geven als parameters en worden veel gebruikt als call back function of event handler.

HO GENT

Delegate: voorbeeld




```
class Program
{
   // declare delegate
   public delegate void Print(int value);

   static void Main(string[] args)
   {
      // Print delegate wijst naar PrintNumber
      Print printDel = PrintNumber;
      // Print printDel = new Print(PrintNumber);

      printDel(100000);
      printDel(200);

      // Print delegate wijst naar PrintMoney
      printDel = PrintMoney;

      printDel(10000);
      printDel(200);
      // printDel.Invoke(200);
   }

   public static void PrintNumber(int num)
   {
      Console.WriteLine("Number: {0,-
12:N0}",num);
   }

   public static void PrintMoney(int money)
   {
      Console.WriteLine("Money: {0:C}", money);
   }
}
```

Een delegate kan opgeroepen worden als een method: een delegate oproepen houdt in de method oproepen waarnaar deze verwijst. Alternatief: Invoke().
Een delegate kan op twee manieren geinstantieerd worden:

- Door de method naam onmiddellijk toe te kennen
- Door een instantie aan te maken van het delegate type met keyword new en als eerste parameter de method naam mee te geven

HO GENT

Delegate als parameter

- Een method kan een parameter hebben van type delegate en kan deze parameter oproepen:

```
public static void PrintHelper(Print delegateFunc, int numToPrint)
{
    delegateFunc(numToPrint);
}
```

HO GENT

Delegate als parameter: voorbeeld

```
class Program
{
    public delegate void Print(int value);

    static void Main(string[] args)
    {
        PrintHelper(PrintNumber, 10000);
        PrintHelper(PrintMoney, 10000);
    }

    public static void PrintHelper(Print delegateFunc, int numToPrint)
    {
        delegateFunc(numToPrint);
    }

    public static void PrintNumber(int num)
    {
        Console.WriteLine("Number: {0,-12:N0}",num);
    }

    public static void PrintMoney(int money)
    {
        Console.WriteLine("Money: {0:C}", money);
    }
}
```

```
Number: 10,000
Money: $ 10,000.00
```

HO GENT

Multicast delegate

- Een delegate kan een referentie bevatten naar meer dan een enkele method:
  - De “+” operator voegt een method toe aan een delegate
  - De “-” operator verwijdert een method uit de set van methods waarnaar een delegate verwijst
- Wanneer een multicast delegate wordt opgeroepen, worden alle methods opgeroepen waarnaar deze delegate verwijst (altijd “en”, nooit “of”) en dit sequentieel

HO
GENT

Multicast delegate: voorbeeld

```
public delegate void Print(int value);

static void Main(string[] args)
{
    Print printDel = PrintNumber;
    printDel += PrintHexadecimal;
    printDel += PrintMoney;

    printDel(1000);

    printDel -=PrintHexadecimal;
    printDel(2000);
}

public static void PrintNumber(int num)
{
    Console.WriteLine("Number: {0,-12:N0}",num);
}

public static void PrintMoney(int money)
{
    Console.WriteLine("Money: {0:C}", money);
}

public static void PrintHexadecimal(int dec)
{
    Console.WriteLine("Hexadecimal: {0:X}", dec);
}
```

```
Number: 1,000
Hexadecimal: 3EB
Money: $ 1,000.00
Number: 2,000
Money: $2,000.00
```

HO GENT

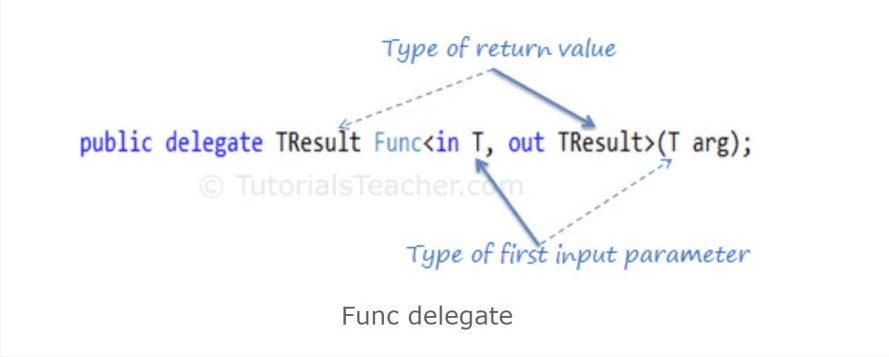
DataProcessor voorbeeld

We maken gebruik van delegates om de berekeningsmethode als een functie mee te geven. We starten met een klasse DataProcessor die een lijst van integer waarden bevat. Deze klasse kan de waarden afdrukken, maar ook een bewerking uitvoeren op de waarden. Welke de bewerking is, maakt geen deel uit van de klasse: we stellen de bewerking in via de methode SetMethod() die een delegate als parameter heeft.
De delegate moet een methode zijn die een lijst van integers als parameter meekrijgt en een double als Resultaat teruggeeft.

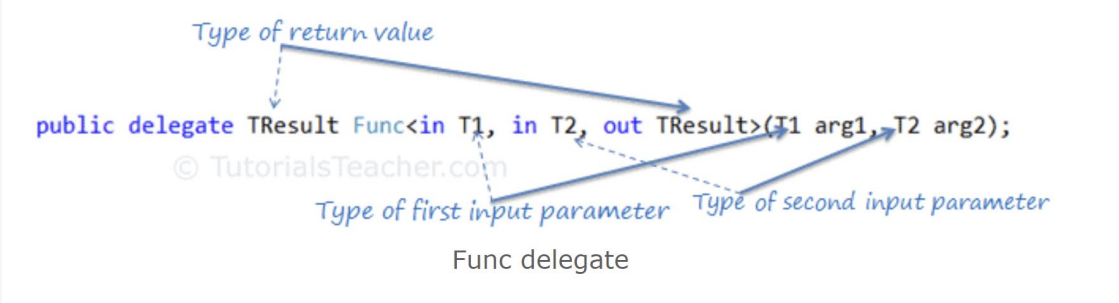
HO
GENT

Func delegate

- Func is een generieke delegate in de System namespace en heeft 0 of meer input parameters en 1 output parameter. De laatste parameter wordt beschouwd als output parameter.
- Lambda expression kan toegekend aan Func<in T, out TResult> delegate



Func delegate



Func delegate

HO GENT

Func delegate: voorbeeld

```
class Program
{
    1 reference | 0 changes | 0 authors, 0 changes
    static int Sum(int x, int y)
    {
        return x + y;
    }

    0 references | 0 changes | 0 authors, 0 changes
    static void Main(string[] args)
    {
        Func<int, int, int> add = Sum;
        var result = add(10, 10);
        Console.WriteLine("Func add result: " + result);
    }
}
```

HO GENT

Func delegate: te onthouden

1. Func is een ingebouwd generiek delegate type
2. Een Func delegate type moet een waarde teruggeven (ALTIJD laatste parameter)
3. Een Func delegate type kan 0 tot 16 input parameters hebben
4. Een Func delegate laat geen ref of out parameters toe
5. Een Func delegate type kan gebruikt worden met een anonieme methode of een lambda uitdrukking

HO
GENT

Action delegate

- Action is eveneens een generiek delegate type gedefinieerd in de System namespace, maar geeft in tegenstelling tot Func **geen waarde terug** (vergelijk met een methode die void als return type heeft).

```
// Action<>: Func<> zonder return type
{
    Action<int> printActionDel = ConsolePrint; // Action<int> printActionDel = new Action<int>(ConsolePrint);
    printActionDel(10);
}
```

```
// Action met een anonieme methode:
{
    Action<int> printActionDel = delegate (int i) { Console.WriteLine(i); };
    printActionDel(10);
}
```

```
// Action met een lambda expressie:
{
    Action<int> printActionDel = i => Console.WriteLine(i);
    printActionDel(10);
}

{
    Action<Student> printStudentDetail = s => Console.WriteLine("Name: {0}, Age: {1} ", s.Name, s.Age);
    var s = new Student() { Name = "Bill", Age = 21 };
    printStudentDetail(s);//output: Name: Bill, Age: 21
}
```

HO GENT

Action delegate: te onthouden

1. Een Action delegate is vergelijkbaar met een Func delegate, maar geeft niets terug. Het return type is void.
2. Een Action delegate kan 0 tot 16 input parameters hebben.
3. Een Action delegate kan gebruikt worden met anonieme methoden en lambda uitdrukkingen.

HO
GENT

DataProcessor voorbeeld herwerkt

We voegen een klasse Printer toe – we opteren hierbij voor een static klasse. In deze klasse voorzien we twee methoden die de waarden op een specifieke manier afdrukken:

- als lijst van waarden gescheiden door een komma
- als lijst van waarden die telkens op een nieuwe lijn worden afgedrukt.

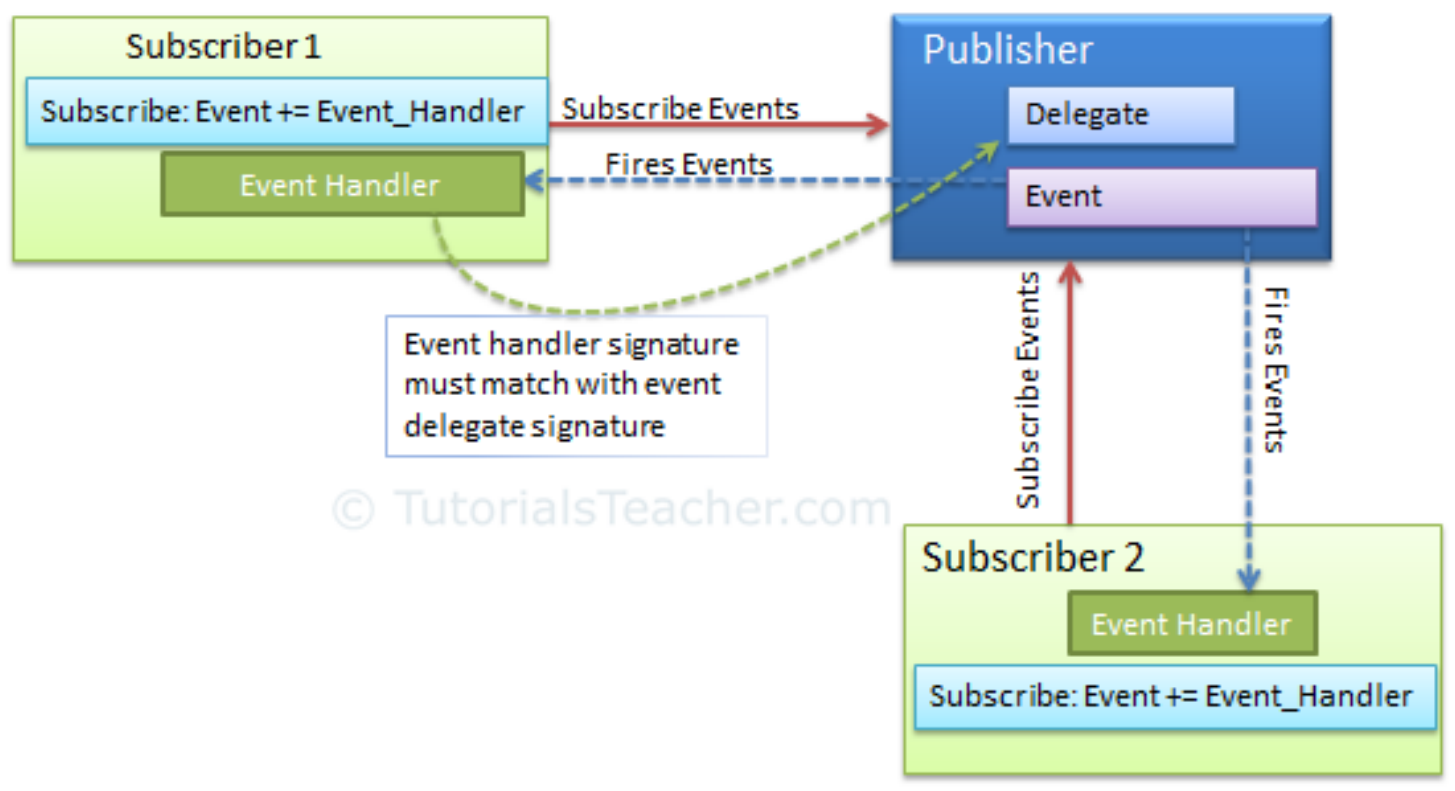
In plaats van delegate ProcessMethod (twee regels) gebruiken we private Func<List<int>, double> _process.

We voegen nog een variabele toe: private Action<List<int>> _printOut; en een public void SetPrint(Action<List<int>> print) {} methode om deze variabele te initialiseren. Methode PrintValues() moet deze Action variabele gebruiken.

HO GENT

Event

- Een “event” is iets wat op het punt staat te gebeuren
- Een “**publisher**” stelt events beschikbaar en lanceert (“raises”) deze:
  - verwittigt dat er een event optreedt (“notification”)
- Een “**subscriber**” schrijft zich in voor een event en onderneemt actie wanneer de notificatie toekomt
- Event = ingepakte (“encapsulated”) delegate
- UI gebruikt vaak events



HO GENT

Event: publisher

```
public class PrintHelper
{
    public delegate void BeforePrint(string message);
    public event BeforePrint beforePrintEvent;

    public void PrintNumber(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent.Invoke("PrintNumber");
        Console.WriteLine("Number: {0,-12:N0}", num);
    }

    public void PrintDecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintDecimal");
        Console.WriteLine("Decimal: {0:G}", dec);
    }

    public void PrintMoney(int money)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintMoney");
        Console.WriteLine("Money: {0:C}", money);
    }

    public void PrintTemperature(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintTemerature");
        Console.WriteLine("Temperature: {0,4:N1} F", num);
    }

    public void PrintHexadecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintHexadecimal");
        Console.WriteLine("Hexadecimal: {0:X}", dec);
    }
}
```

HO GENT

Event: subscriber

```
class Number
{
    private PrintHelper _printHelper;

    public Number(int val)
    {
        _value = val;
        _printHelper = new PrintHelper();
        _printHelper.beforePrintEvent += printHelper_beforePrintEvent;
    }

    void printHelper_beforePrintEvent(string message)
    {
        Console.WriteLine("BeforePrintEvent fires from {0}",message);
    }
    private int _value;

    public int Value
    {
        get { return _value; }
        set { _value = value; }
    }

    public void PrintMoney()
    {
        _printHelper.PrintMoney(_value);
    }

    public void PrintNumber()
    {
        _printHelper.PrintNumber(_value);
    }
}
```

```
BeforePrintEvent fires from PrintMoney.
Money: $ 1,00,000.00
BeforePrintEvent fires from PrintNumber.
Number: 1,00,000
```

HO GENT

C# Events: wat leerden we?

- Gebruik event keyword met delegate type om een event te declareren
- Controleer of event null is of niet vooraleer je een event oproept
- Schrijf in op events met += en schrijf uit met -=
- Event handler heeft zelfde signatuur als event delegate
- Een klasse/interface kan een event als “member” hebben

HO
GENT

.NET Framework EventHandler

- .NET framework gebruikt de generieke EventHandler delegate en de EventArgs base class; WPF (grafische interfaces) maakt hiervan bijvoorbeeld veel gebruik.

⇒ Wij kunnen deze techniek ook steeds gebruiken!

- Zie voorbeeld KlantOberKok en bouw dit na:
  - https://www.youtube.com/watch?v=b_RyVn56r5s&list=PLM3q9wWBZWb90CajLrVZcenmxIBCqyaFq&index=3&t=681s
  - https://www.youtube.com/watch?v=RcTLxJ6FIZQ&list=PLM3q9wWBZWb90CajLrVZcenmxIBCqyaFq&index=4

HO GENT

Extension method

- Zoals de naam suggereert: een bijkomende *method*
- Injecteer een method zonder een *class/struct/interface* te wijzigen, af te leiden of hercompileren
- Kan toegevoegd worden aan een *class* van jezelf, het .NET framework of andere partijen
- Een speciale *static method* in een *static class*

HO
GENT

Extension method: voorbeeld 1

In een eerste eenvoudige voorbeeld maken we een extension method die toelaat om string terug te geven waar we zowel voor-als achteraan een extra aantal tekens toevoegen. De extension method brengen we onder in een static class en definiëren de methode zelf ook als static. Verder is het belangrijk om de eerste parameter in de methode te definiëren als het uit te breiden type (in ons geval string) en dit te laten voorafgaan met het keyword ‘this’.

HO
GENT

Extension method: voorbeeld 2

We geven ook nog een tweede voorbeeld.We hebben de klasse Segment die een lijnstuk voorstelt en bestaat uit een lijst van punten. Voor het beschrijven van een punt maken we gebruik van de klasse Point die als properties een x-en y-coördinaat bevat. Met de extension method die we toevoegen, willen we de lengte van het segment/lijnstuk berekenen. We maken dus een static klasse SegmentExtension aan met daarin een static methode Length die een double zal teruggeven en als parameter enkel het segment heeft (let op keyword this). Voor het berekenen van de afstand verwijzen we naar https://nl.wikihow.com/De-afstand-tussen-twee-punten-berekenen.

HO
GENT

Extension method: voorbeeld 3

```
namespace ExtensionMethods
{
    public static class IntExtensions
     {
        public static bool IsGreaterThan(this int i, int value)
        {
            return i > value;
        }
    }
}

using ExtensionMethods;

class Program
{
    static void Main(string[] args)
    {
        int i = 10;

        bool result = i.IsGreaterThan(100);

        Console.WriteLine(result);
    }
}
```

HO GENT

Anonymous method

- Een method zonder naam
- Delegate keyword: kan toegekend aan variabele van het delegate type

```
public delegate void Print(int value);

static void Main(string[] args)
{
    Print print = delegate(int val) {
        Console.WriteLine("Inside Anonymous method. Value: {0}", val);
    };

    print(100);
}
```

HO GENT

Anonymous method

- Kan doorgegeven worden aan method met delegate parameter:

```
public delegate void Print(int value);

class Program
{
    public static void PrintHelperMethod(Print printDel, int val)
    {
        val += 10;
        printDel(val);
    }

    static void Main(string[] args)
    {
        PrintHelperMethod(delegate(int val) { Console.WriteLine("Anonymous method: {0}", val); }, 100);
    }
}
```

Anonymous method

- Kan variabelen gebruiken van omsluitende method
- Kan gebruikt worden als event handler
- Niet ondersteund: goto, break, continue, ref of out parameter van omsluitende method

```
public delegate void Print(int value);

static void Main(string[] args)
{
    int i = 10;

    Print prnt = delegate(int val)
    {
        val += i;
        Console.WriteLine("Anonymous method: {0}", val);
    };

    prnt(100);
}
```

HO GENT

C# 3.0: lambda expression

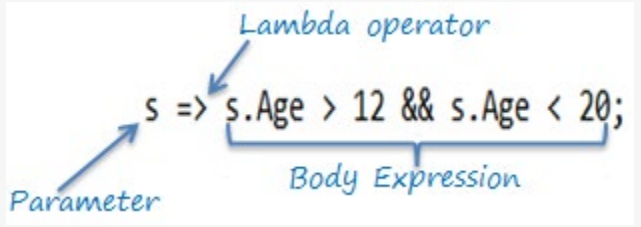
- Een lambda uitdrukking is een kortere manier om een anonieme methode te schrijven
- Voorbeeld:

```
delegate(Student s) { return s.Age > 12 && s.Age < 20; };
```

1 - Remove Delegate and Parameter Type and add lamda operator =>

```
delegate(Student s) => { return s.Age > 12 && s.Age < 20; };
```

```
(s) => { return s.Age > 12 && s.Age < 20; };
```




```
(s) => { return s.Age > 12 && s.Age < 20; };
```

2 - Remove curly bracket, return and semicolon

```
(s) => s.Age > 12 && s.Age < 20;
```

3 - Remove Parenthesis around parameter if there is only one parameter

```
s => s.Age > 12 && s.Age < 20;
```


HO GENT

C# 3.0: lambda zonder parameter

```
() => Console.WriteLine("Parameter less lambda expression")
```

HO
GENT

C# 3.0: lambda multiple parameters

```
(s, youngAge) => s.Age >= youngAge;
```

HO
GENT

C# 3.0: lambda multiple statements

```
(s, youngAge) =>
{
  Console.WriteLine("Lambda expression with multiple statements in the body");

  Return s.Age >= youngAge;
}
```

HO
GENT

C# 3.0: lambda – wat leerden we?

1. Een lambda uitdrukking laat toe een anonieme methode korter te noteren
2. Syntax: parameters => body expression
3. Een lambda uitdrukking kan 0 parameters hebben
4. Een lambda uitdrukking kan meer dan 1 parameter hebben tussen haakjes (), gescheiden door een komma
5. Een lambda uitdrukking kan meer dan 1 statement hebben in de body expression, tussen accolades {}
6. Een lambda uitdrukking kan toegekend worden aan Func, Action, Predicate (delegate types)
7. Een lambda uitdrukking kan opgeroepen worden als een delegate

HO
GENT

Func delegate: voorbeeld

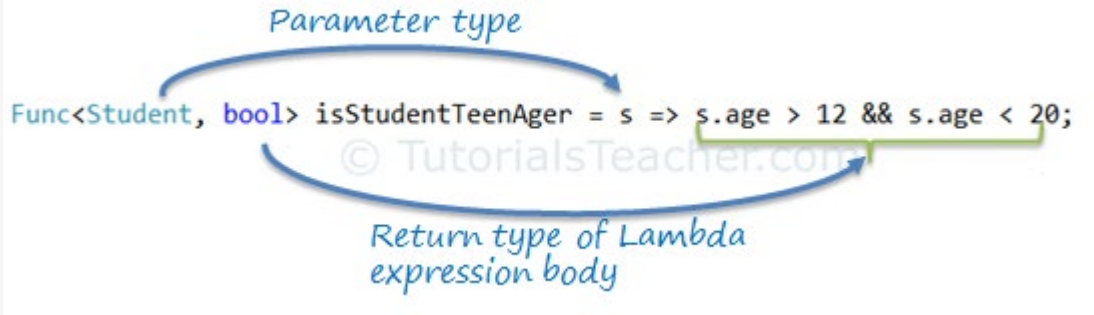
```
// Func met een anonieme methode:
{
    Func<int> getRandomNumber = delegate () { var rnd = new Random(); return rnd.Next(1, 100); };
    var result = getRandomNumber();
    Console.WriteLine("Func delegate result: " + result);
}
```

```
// Func met een lambda expressie:
{
    Func<int> getRandomNumber = () => new Random().Next(1, 100);
    var result = getRandomNumber();
    Console.WriteLine("Func lambda result: " + result);
    Func<int, int, int> sum = (x, y) => x + y;
    result = sum(5, 6);
    Console.WriteLine("Func lambda sum result: " + result);
}
```

HO GENT

Func delegate

- Lambda expression kan toegekend aan Func<in T, out TResult> delegate:




```
{
    Func<Student, bool> isStudentTeenAger = s => s.Age > 12 && s.Age < 20;
    var s = new Student() { Age = 21 };
    var isTeenAger = isStudentTeenAger(s);
    Console.WriteLine("Student is teenager: " + (isTeenAger ? "yes" : "no"));
}
```

HO GENT