

(T24)討論 FuncDelegate(委派)、LamdaExpression(表達式)、AnonymousMethod(匿名方法)  
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92

---

(T24)討論 FuncDelegate(委派)、LamdaExpression(表達式)、AnonymousMethod(匿名方法)

---

## 0. Student Questions

### 1. New Project

#### 1.1. Create New Project : Sample

### 2. Sample : Program.cs

---

## 0. Student Questions

學生提問

<https://www.facebook.com/groups/934567793358849/posts/2040629736085977/>

老師及前輩們好

小弟有課程上關於 Predicate、Fun、以及 Enumerable 的問題想請教，先前有發過文，這裡再依指定格式重新發文一次

分述如下：

Tutorial 024 完全理解 FuncDelegate 委派和 LamdaExpression 表達式和 AnonymousMethods 匿名方法之中

影片連結：

<https://hiskio.com/courses/182/lectures/7142>

講義連結：

<https://ithandyguytutorial.blogspot.com/2017/12/t024funcdelegatelambdaexpressionanonymo.html>

影片 05:11 處，講義 1. Step 3 之中

```
Gamer gameId1 = listGamers.Find(g => predicateGetGamerId1(g));
```

此處 g 為什麼會代表 Gamer?原理是什麼?

g => predicateGetGamerId1(g)所執行的邏輯是將每一個 listGamers 的元素 Gamer 作為參數，呼叫 point method◇GetGamerId1，然後分別傳回 bool 值嗎?

同章節影片 09:06 處，講義 1. Step 3 之中

```
Gamer gameId3V2 = listGamers.Find(g => g.Id == 3);
```

此處 g 為什麼會代表 Gamer?原理是什麼?

同章節影片 11:38 處，講義 2.1.之中

```
Func<Gamer, string> funcDelegateSelector = employee => $"Name == {employee.Name}";
```

其中 Gamer 是 input parameter 的型別，string 是 ouput 返回值的型別，這邊設計設計上我不懂，是藉由 Func <Gamer,string>

來設定 employee => \$"Name == {employee.Name}"的傳入參數 employee 的型別，以及 \$"Name == {employee.Name}"的返回型別嗎?

另外，Func 和 Predicate 的主要差別是什麼?

---

LINQ 完全攻略(C#)中

Tutorial02 linQ 總計中

影片連結：

<https://hiskio.com/courses/181/lectures/7012>

講義連結:

<https://ithandyguytutoria.blogspot.com/2017/12/t002linqaggregateminmaxsumcountaverage.html>

影片 09:13，講義 2.Using Lambda Expressions.之中

```
IEnumerable<GamerA> allFemaleV2 = listGamerA.Where(gamer => gamer.Gender == "Female");
```

其中 Where 裡面要放 Func，請問本例中 Func<Gamer,bool>是如何產生的?因為上述 Where 的參數只有 gamer => gamer.Gender == "Female"，我不明白是什麼設定而導致 Func<Gamer,bool>這個結果

Tutorial03 完整攻略 Linq to Object 的 Where 搜尋語法中

影片連結:

<https://hiskio.com/courses/181/lectures/7015>

講義連結:

<https://ithandyguytutoria.blogspot.com/2017/12/t003where.html>

影片 04:45，講義 1.中提到的

```
Enumerable.Where<TSource>(this
```

```
IEnumerable<TSource> source, Func<TSource, Boolean> filter)
```

其中 Where 有 this IEnumerable<TSource> source 這個參數，但案例 IEnumerable<int> intOddV1 = intList.Where(num => IsOdd(num));

卻沒看到這個參數，似乎僅有 Func<TSource, Boolean> filter

this IEnumerable<TSource> source 這個參數是什麼?有什麼意義呢?使用上哪時候要填，哪時候不需要填?

影片 05:15 中提到的 extend method，是什麼?

感謝!

接下來是強者 Sam Chuang 的回答

問：

```
Gamer gameId1 = listGamers.Find(g => predicateGetGameId1(g));
```

此處 g 為什麼會代表 Gamer?原理是什麼?

g => predicateGetGameId1(g)所執行的邏輯是將每一個 listGamers 的元素 Gamer 作為參數，呼叫 point method GetGameId1，然後分別傳回 bool 值嗎?

答：

在回答 g 為什麼會代表 Gamer 之前，要先理解一下 Find() 方法是什麼內容

透過 ILSpy 反組譯後，可看到他是 List<T> 提供的方法，原型如下

```
public T Find(Predicate<T> match)
{
    if (match == null)
    {
        ThrowHelper.ThrowArgumentNullException(ExceptionArgument.match);
    }
    for (int i = 0; i < _size; i++)
    {
        if (match(_items[i]))
        {
            return _items[i];
        }
    }
    return default(T);
}
```

從範例 listGamers 得知 T 就是 Gamer，帶入後就是

```
public Gamer Find(Predicate<Gamer> match)
```

參數 Predicate<Gamer> match 可看成如下方法

bool 方法名稱不重要 (Gamer 參數名稱不重要)

```
{
    // 委派填入的方法內容
}
```

傳入 match 的 g => predicateGetGameId1(g) 是 lambda 表達式的簡化寫法

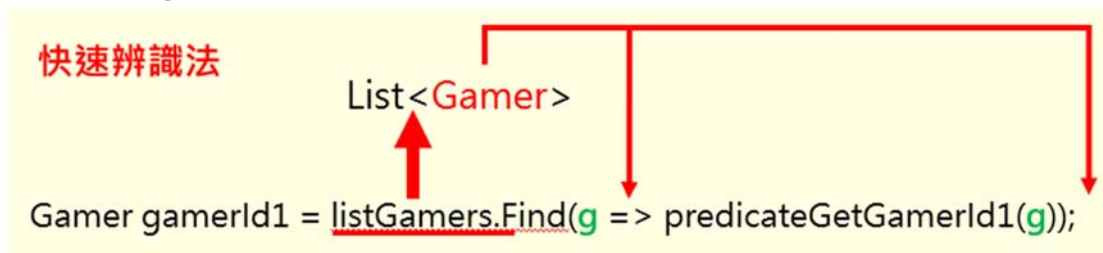
和 `Predicate<Gamer>` 組合可得到如下的完整寫法

`bool` 方法名稱不重要 (`Gamer g`)

```
{  
    return predicateGetGamerId1(g);  
}
```

由此可知 `g` 就是 `Gamer`

不一定要叫 `g`，你想取任何名稱都可以



Kevin 補充

```
Gamer gamerId3V2 = listGamers.Find(g => g.Id == 3)
```

其中

```
g => g.Id == 3
```

這部分其實只是一個 `Anonymous method`(匿名方法)

這 `method` 如果要完整地寫出來大約如下

```
bool AAAA(Gamer g) {  
    return g.Id == 3  
}
```

也就是說這裡的 `g` 其實只是一個參數名稱

也可以改稱 `lol`(誰沒玩過，顆顆)

```
Gamer gamerId3V2 = listGamers.Find(lol => lol.Id == 3)
```

只是因為這個 `method` 的內容只有一行

所以我們沒有必要寫整個 `method` 出來

就變成 `Anonymous method`(匿名方法)

接下來是強者 Sam Chuang 的回答

問：

```
Func<Gamer, string> funcDelegateSelector = employee => $"Name == {employee.Name}";
```

其中 `Gamer` 是 `input parameter` 的型別，`string` 是 `ouput` 返回值的型別，這邊設計設計上我不懂，

是藉由 `Func <Gamer,string>` 來設定 `employee => $"Name == {employee.Name}"` 的傳入參數 `employee` 的型別，以及 `"Name == {employee.Name}"` 的返回型別嗎？

另外，`Func` 和 `Predicate` 的主要差別是什麼？

答：

你可以用 `delegate` 去建立自訂方法的委派

微軟也包裝了幾種常用的委派 `Func`、`Predicate`、`Action` 說明如下

### **Func<TResult> 用在會回傳指定型別的方法**

泛型最多可以傳入 9 個型別，前 8 個是傳入方法參數的型別，最後一個是方法回傳的型別

例如 `Func<int, string, bool, 自訂類別, ..., 回傳的指定型別> delegateFunc = ...;`

舉例傳入 2 個型別的原型如下

```
public delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2);
```

### **Predicate<T> 用在回傳 bool 的方法**

原型如下

```
public delegate bool Predicate<in T>(T obj);
```

---

### Action<T> 用在 void 不會有回傳的方法

泛型最多可以傳入 8 個型別

例如 Action<int, string, bool, 自訂類別, ....> delegateAction = ....;

舉例傳入 2 個型別的原型如下

```
public delegate void Action<in T1, in T2>(T1 arg1, T2 arg2);
```

---

回到原本的問題 employee => \$"Name == {employee.Name}" 是 lambda 表示式

如果寫成方法的話大概長這樣

```
string GetName(Gamer employee)
```

```
{  
    return $"Name == {employee.Name}";  
}
```

再把這個方法指派給 Func<Gamer, string> funcDelegateSelector 這個委派

等同以下寫法

```
void Main()  
{  
    myDelegate funcDelegateSelector = new myDelegate(GetName);  
}  
delegate string myDelegate(Gamer employee);
```

---

接下來是強者 Sam Chuang 的回答

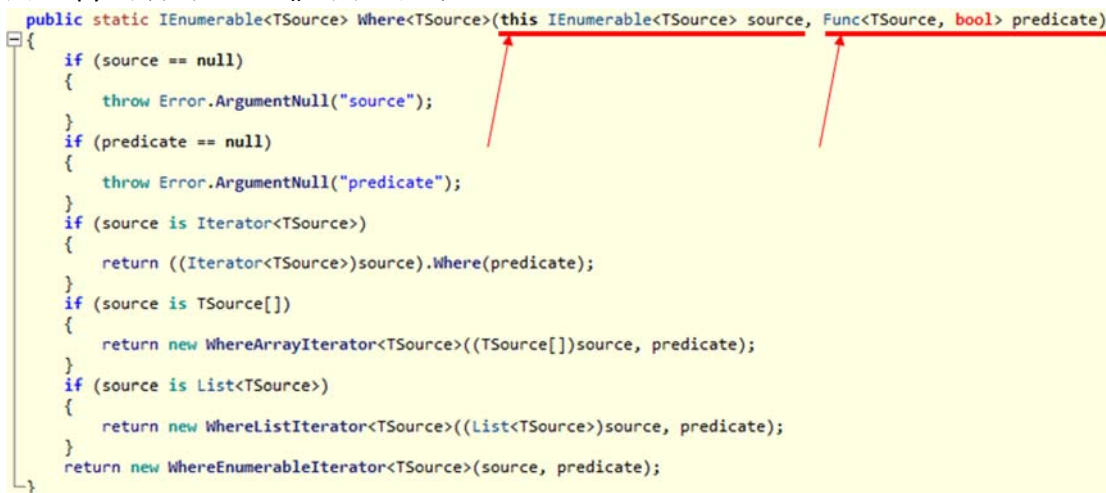
問：

```
IEnumerable<GamerA> allFemaleV2 = listGamerA.Where(gamer => gamer.Gender == "Female");
```

其中 Where 裡面要放 Func，請問本例中 Func<Gamer,bool>是如何產生的?因為上述 Where 的參數只有 gamer => gamer.Gender == "Female"，我不明白是什麼設定而導致 Func<Gamer,bool>這個結果

答：

用 ILSpy 可看到 Where() 的原型如下



```
public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate)  
{  
    if (source == null)  
    {  
        throw Error.ArgumentNull("source");  
    }  
    if (predicate == null)  
    {  
        throw Error.ArgumentNull("predicate");  
    }  
    if (source is Iterator<TSource>)  
    {  
        return ((Iterator<TSource>)source).Where(predicate);  
    }  
    if (source is TSource[])  
    {  
        return new WhereArrayIterator<TSource>((TSource[])source, predicate);  
    }  
    if (source is List<TSource>)  
    {  
        return new WhereListIterator<TSource>((List<TSource>)source, predicate);  
    }  
    return new WhereEnumerableIterator<TSource>(source, predicate);  
}
```

第一個參數表示 Where 是一個擴充方法

第二個參數表示需要傳入一個 Func<TSource, bool> 答案就在此

---

接下來是強者 Sam Chuang 的回答

問：

```
Enumerable.Where<TSource>(this IEnumerable<TSource> source, Func<TSource, Boolean> filter)
```

其中 Where 有 this IEnumerable<TSource> source 這個參數，但案例 IEnumerable<int> intOddV1 = intList.Where(num => IsOdd(num));

卻沒看到這個參數，似乎僅有 Func<TSource, Boolean> filter

this IEnumerable<TSource> source 這個參數是什麼?有什麼意義呢?使用上哪時候要填，哪時候不需要填?

影片 05:15 中提到的 extend method，是什麼？

答：

第一個參數 this 表示這是一個擴充方法，請 Google "C# Extension Methods"

Kevin 補充

請參考

Tutorial31 講義: 完全攻略 Extend Method 擴充方法

=====

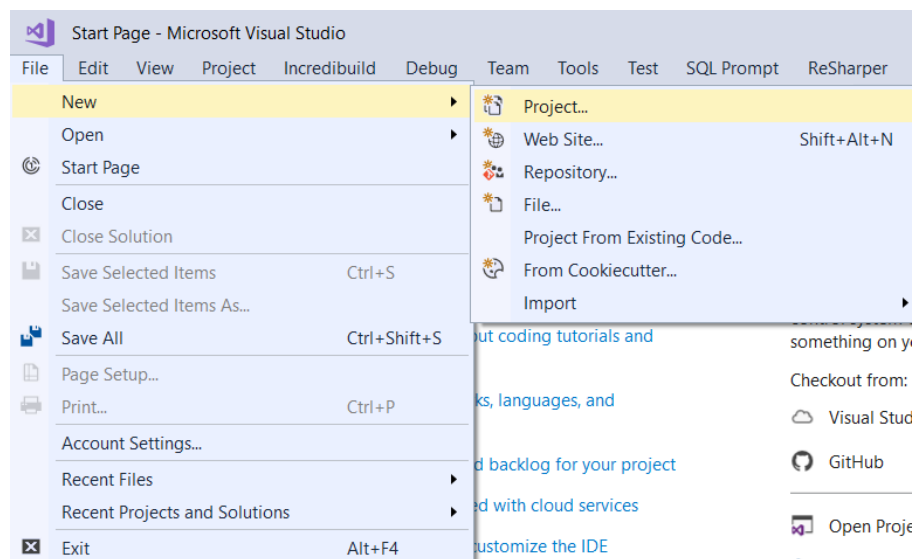
# 1. New Project

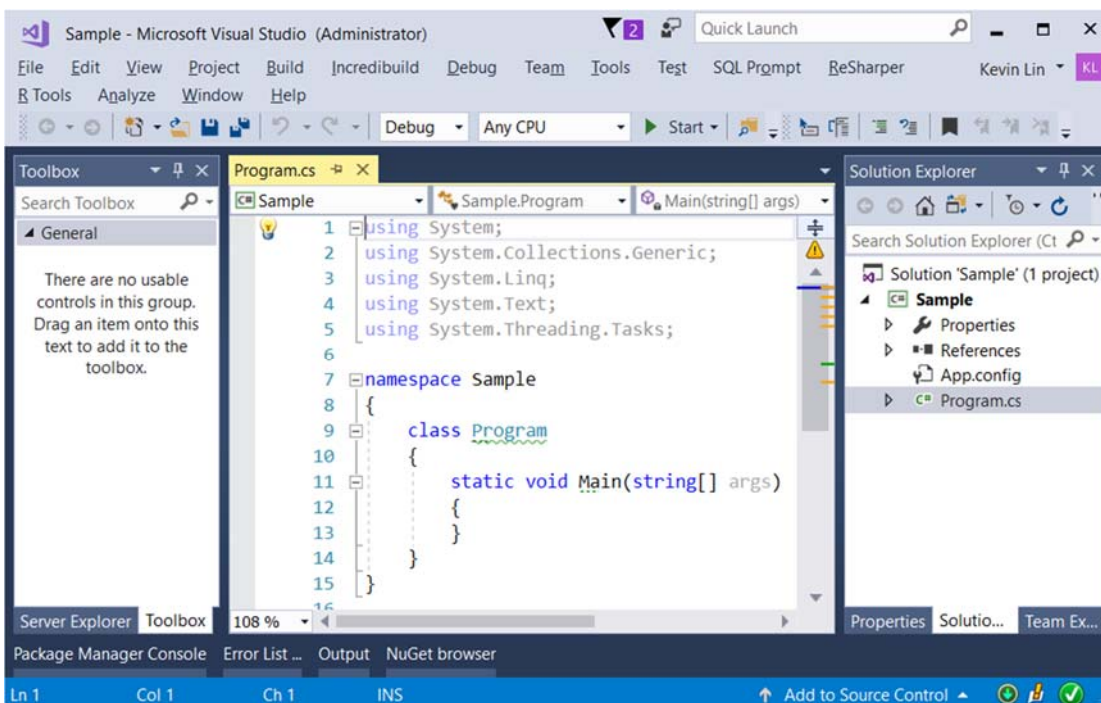
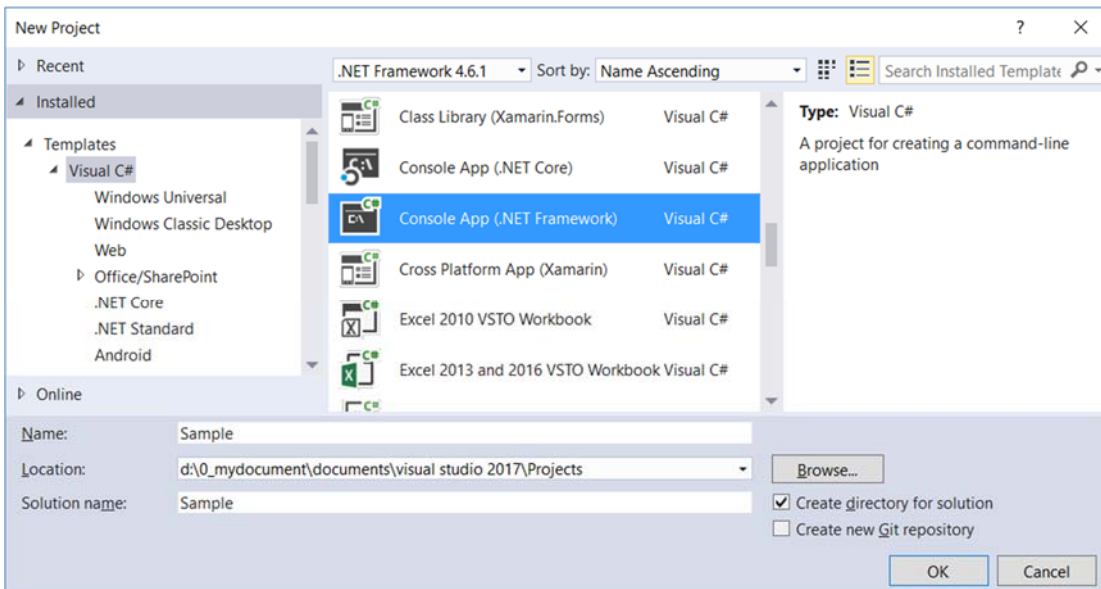
## 1.1. Create New Project : Sample

File --> New --> Project... -->

Visual C# --> **Console App (.Net Framework)** -->

Name: **Sample**





=====

## 2. Sample : Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using OnlineGame;
namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // 1. =====
            //Anonymous methods
```



```

Console.WriteLine("1. AnonymousMethodsSample() ===== ");
AnonymousMethodsSample();
// 2. =====
//Func<T, TResult> Delegate
Console.WriteLine("2. FuncDelegateSample() ===== ");
FuncDelegateSample();
Console.ReadLine();
}
//1. =====
//Anonymous methods
static void AnonymousMethodsSample()
{
    List<Gamer> listGamers = new List<Gamer>
    {
        new Gamer{ Id = 1, Name = "Name01"},
        new Gamer{ Id = 2, Name = "Name02"},
        new Gamer{ Id = 3, Name = "Name03"},
        new Gamer{ Id = 4, Name = "Name04"}
    };
    //Step 2: Create a Predicate<Gamer> delegate object
    //with GetGamerId1 method as parameter.
    Predicate<Gamer> predicateGetGamerId1 = new Predicate<Gamer>(GetGamerId1);
    //Step 3: pass the delegate instance as parameter of Find()
    Gamer gamerId1 = listGamers.Find(g => predicateGetGamerId1(g));
    Console.WriteLine($"predicateGetGamerId1 : gamerId1.Id=={gamerId1.Id},
gamerId1.Name=={gamerId1.Name}.");
    // "new Predicate<Gamer> " can be omitted.
    Predicate<Gamer> predicateGetGamerId2 = GetGamerId2;
    Gamer gamerId2 = listGamers.Find(g => predicateGetGamerId2(g));
    Console.WriteLine($"predicateGetGamerId2 : gamerId2.Id=={gamerId2.Id},
gamerId2.Name=={gamerId2.Name}.");
    // Anonymous method is being passed as an argument to
    // the Find() method. This anonymous method replaces
    // the need for Step 1, 2 and 3
    Gamer gamerId3 = listGamers.Find(delegate (Gamer g) { return g.Id == 3; });
    Console.WriteLine($"predicateGetGamerId3 : gamerId3.Id=={gamerId3.Id},
gamerId3.Name=={gamerId3.Name}.");
    //using lambda expression
    //=> is called lambda operator and read as GOES TO
    Gamer gamerId3V2 = listGamers.Find(g => g.Id == 3);
    Console.WriteLine($"predicateGetgamerId3V2 : gamerId3V2.Id=={gamerId3V2.Id},
gamerId3V2.Name=={gamerId3V2.Name}.");
    //using lambda expression
    Gamer gamerId3V3 = listGamers.Find((Gamer g) => g.Id == 3);
    Console.WriteLine($"predicateGetgamerId3V2 : gamerId3V3.Id=={gamerId3V3.Id},
gamerId3V3.Name=={gamerId3V3.Name}.");
}
// Step 1: Create a method whose signature matches Predicate<Gamer> delegate.
private static bool GetGamerId1(Gamer g)
{
    return g.Id == 1;
}
private static bool GetGamerId2(Gamer g)
{
    return g.Id == 2;
}

```

```

//2. =====
//Func<T, TResult> Delegate
static void FuncDelegateSample()
{
    List<Gamer> listGamers = new List<Gamer>
    {
        new Gamer{ Id = 1, Name = "Name01"},
        new Gamer{ Id = 2, Name = "Name02"},
        new Gamer{ Id = 3, Name = "Name03"},
        new Gamer{ Id = 4, Name = "Name04"}
    };
    //2.1. -----
    // Create Func<T, TResult> Delegate
    // and pass it to the Select() LINQ function,
    // ListObject.Select(funcDelegate)
    Console.WriteLine("2.1. Func<T, TResult> Delegate -----");
    Func<Gamer, string> funcDelegateSelector =
        employee => $"Name == {employee.Name}";
    IEnumerable<string> names = listGamers.Select(funcDelegateSelector);
    foreach (string nameItem in names)
    {
        Console.WriteLine(nameItem);
    }
    // 2.2. -----
    // lambda expression
    Console.WriteLine("2.2. lambda expression -----");
    IEnumerable<string> names2 =
        listGamers.Select(employee => "Name == " + employee.Name);
    foreach (string nameItem in names2)
    {
        Console.WriteLine(nameItem);
    }
    // 2.3. -----
    // Create Func<T, T, TResult> Delegate
    Console.WriteLine("2.3. Func<T, T, TResult> Delegate -----");
    Func<int, int, string> funcDelegateSelector2 =
        (i1, i2) =>
            "Sum == " + (i1 + i2).ToString();
    string result = funcDelegateSelector2(10, 20);
    Console.WriteLine(result);
}
}
namespace OnlineGame
{
    public class Gamer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public override string ToString()
        {
            return $"Id=={Id} ; Name=={Name}";
        }
    }
}

```



```
1. AnonymousMethodsSample() =====
predicateGetGamerId1 : gameId1.Id==1, gameId1.Name==Name01.
predicateGetGamerId2 : gameId2.Id==2, gameId2.Name==Name02.
predicateGetGamerId3 : gameId3.Id==3, gameId3.Name==Name03.
predicateGetgamerId3V2 : gameId3V2.Id==3, gameId3V2.Name==Name03.
predicateGetgamerId3V2 : gameId3V3.Id==3, gameId3V3.Name==Name03.
2. FuncDelegateSample() =====
2.1. Func<T, TResult> Delegate -----
Name == Name01
Name == Name02
Name == Name03
Name == Name04
2.2. lambda expression -----
Name == Name01
Name == Name02
Name == Name03
Name == Name04
2.3. Func<T, T, TResult> Delegate -----
Sum == 30
```