## Anonymous Methods

מתודות אנונימיות מאפשרות כתיבת מתודות בלי שם.

עד עכשיו כשעבדנו עם delegates הגדרנו מתודה, ואז הצבענו עליה כשיצרנו מופע. (בתהליך האינסטנטינציה - Instantiation). כלומר, העברנו כפרמטר את שם הפונקציה בה השתמשנו כשהגדרנו את המופע של ה delegate (בשלב intantiation 2).

לדוגמה:

```
// Declare a delegate.
        delegate void Del(int x);

        // Define a named method.
        void DoWork(int k) { /* ... */ }

        // Instantiate the delegate using the method as a parameter.
        Del d = obj.DoWork;
```

החל מגרסה 2 של #C מתאפשר ליצור מופעים של delegates, ומיד להגדיר את קטע הקוד שיורץ כשיקראו לפונקציה (כשיעשו invocation). זה פשוט מקצר לנו את הקוד וחוסך לנו הגדרה בנפרד של המתודה.

```
// Declare a delegate.
delegate void Del(string name);


  // Instantiate Del by using an anonymous method.
Del del3 = delegate (string name)
                        { Console.WriteLine($"Notification received for: {name}"); };
```

שימו לב, שבדוגמה הזו בהגדרת המופע מיד כתבתי בלוק קוד להפעלה עליה אני מצביע ולא העברתי שם של מתודה שהגדרתי במקום אחר. אני משתמש בזה כאשר מיותר לי להגדיר מתודה בנפרד, ואני צריך רק את המצביע למתודה.

להגרת מתודה אנונימית, משתמשים במילת המפתח delegate (אותה מילת מפתח בה אנו משתמשים להגדרת ה delegate), ואז בסוגריים את הפרמטרים שלנו ואת בלוק הקוד.

ועוד דוגמה:

```
MyFirstDelegate first1 = delegate () { Console.WriteLine("Shalom"); };
MyFirstDelegate first2 = delegate () { Console.WriteLine("Another function called
with same delegate"); };
```

בשביל להבין מתי נשתמש בזה "בחיים האמיתיים" ניזכר במתודות כמו Sort או Where שהוגדרו במחלקה List. אנחנו צריכים להעביר לה מתודה שעל פי המתודה הזו מתבצע ה Sort. במקום להגדיר את המתודה בנפרד ולהעביר אותה, הרבה יותר נוח במהלך הכתיבה שלנו, פשוט לכתוב את המתודה שמגדירה את הקריטריון של ה Sort או הפילטר.

## Lambda Expressions

משתמשים בביטויי Lambda בכדי ליצור פונקציה אנונימית.
יכולת שהגיעה ב #C 3 לשפה, ומקצרת את התחביר עוד יותר מאשר ב Anonymous Methods (שהגיעו עם #C 2).

ביטויי למבדה מקובלים היום כתחביר מקוצר לפונקציות במגוון של שפות בתעשייה.

ליציר ביטוי למבדה, נשתמש באופרטור `=>` בכדי להפריד בין רשימת הפרמטרים של הפונקציה לגוף הפונקציה. נקרא לזה מעתה אופרטור הלמבדה.
הפונקציות שנכתוב בתחביר Lambda מתחלקות לשניים:

1. ביטויי למבדה, המכילות רק ביטוי שמחזיר ערך, והמבנה שלהן הוא:

```
(input - parameters) => expression
```

2. משפטי למבדה, או פקודות למבדה, בהן גוף הפונקציה, הוא בלוק של קוד:

```
(input-parameters) => { <sequence-of-statements> }
```

בכדי ליצור ביטוי למבדה, נרשום בצד שמאל של אופרטור הלמבדה, את רשימת הפרמטרים (אם ישנם) ובצד ימין ביטוי או בלוק של קוד.

ניתן להמיר כל ביטוי למבדה ל delegate. סוג ה delegate נקבע על פי הפרמטרים שפונקציית הלמבדה הגדירה והערך המוחזר שלה.
בואו נתבונן בדוגמה הבאה:

```
delegate void Greet(string name);
Greet greet3 = (name)=> { Console.WriteLine("Shalom {0}", name); };
Greet greet4 = (name)=> { Console.WriteLine("Welcome {0}", name); };
```

הגדרנו delegate בשם Greet עם חתימה מסויימת. אנחנו מגדירים פונקציה אנונימית מקומית ומצביעים עליה עם משתנים מסוג ה delegate שהגדרנו. התחביר של הפונקציה קצרצר.

ודוגמה לביטוי למבדה:

```
delegate int MathDelegate (int x);
MathDelegate Power = number => number * number;
Console.WriteLine(Power(5));//25
```

כפי שאנו רואים, כאשר אנחנו רק רוצים להחזיר ערך, ניתן להשתמש בביטוי, בלי להגדיר סוגריים מסולסלות {} , כלומר, בלי להגדיר בלוק של קוד, ויש לנו ביטוי למבדה.
זה שווה ערך לשורה הבאה:

```
MathDelegate Power = number => { return number* number};
```

ניתן להבחין שבהגדרת הפרמטר number בצד שמאל שלאופרטור הלמבדה, השמטתי את הסוגריים. ניתן להשמיט את הסוגריים של הפרמטרים כאשר יש לנו ביטוי למבדה שמכיל רק פרמטר אחד בודד. אם אין פרמטרים בפונקציה, נכצוב סוגריים ריקות ().
לדוגמה:

```
MyFirstDelegate first4 = () =>
{
      Console.WriteLine("Another function called with same delegate");
};
```

## תרגילי כיתה ובית

לטובת תרגול ניתן לבצע את התרגילים שהיו לנו ל Delegates הפעם עם Annonymus Methods ועם Lambda Expressions.