

התחלנו לדבר על לולאות (קטע פקודות שחוזר על עצמו) מסוג - for

```
for (int j=2; j <10; ++j)
{
    Console.WriteLine( j, j*j, j*j*j);
}
```

אמרנו שהכותרת (משפט ה- for) הוא בעל 3 חלקים.

1. השמת ערך למשתנה (והגדרתו - אפשר גם להגדירו קודם), בדוגמה: j
2. בדיקת התנאי (באמצע), בדוגמה: "כל עוד j קטן מעשר"
3. הגדלת משתנה הלולאה (++j)

הפעולה השלישית מתבצעת לאחר הפקודות שבתוך גוף הלולאה, בדיקת התנאי נעשית לפני ביצוע גוף הלולאה.

אסור לשנות את משתנה הלולאה (j) בתוך הלולאה על ידי פקודה. הוא נישלט עלידי משפט ה- for

הלולאה הנ"ל תדפיס את המספרים מ-2 ועד 9, הריבוע של כל אחד והחזקה השלישית של כל אחד.

כתרגיל נא לכתוב לולאה שמדפיסה את המספרים 20 עד 0, אחורה. כך:

```
20
19
18
17
.
.
.
.
.
0
```

משתנה האינדקס של לולאת ה- for יכול להיות מוגדר גם לפני הלולאה, למשל:

```
int i;
for (i=1; i<4; ++i)
{
}
```

אם משתנה אינדקס מוגדר בתוך הסוגריים של ה- for, הוא 'נעלם' לאחר סיום הלולאה (לא קיים יותר)

אפשר גם להגדיר ולאתחל יותר ממשתנה אינדקס אחד גם להתייחס אליהם ביתר החלקים של משפט ה- for

למשל:

```
for(int i=1, num=10; i<5 && num>5; ++i; num -=2)
{
    Console.WriteLine("Hello");
}
```

כמה פעמים יודפס כאן Hello ?

## לולאה אינסופית (שמוש רווח)

```
for(;;)
{
    string s1=Console.ReadLine();
    if (s1!="a")
    {
        Console.WriteLine("Loop me for ever.... and a day");
    }
    else
        break;
}
```

אם החלק המרכזי של ה- for לא קיים, ההנחה זה שהוא נכון תמיד. לולאות אינסופיות שימושיות עבור משחקים.

כפי שרואים כאן, השמטתי את כל חלקי ה- for והשארתי רק את המפרידים (; נקודה-פסיק) - למרבה הפלא זה חוקי!

תרגיל כיתה

כתוב תוכנית שמדפיסה את 5 המספרים המשוכללים הראשונים. מספר משוכלל זהו מספר שלם, חיובי השווה לסכום כל מחלקיו (כולל 1).

למשל, מספר המשוכלל ראשון הוא: 6 כיוון שהמחלקים שלו הם: 1, 2, 3  $1+2+3=6$   
המספר המשוכלל הבא הוא: 28 (נסו ותיווכחו).

זה אומר שאם כתבתם נכון את התוכנית, שני המספרים הראשונים שיודפסו הם: 6 28

## חזרה למחלקות ועצמים

עד היום, כשרצינו לפתור בעיה, פיתחו שיטת עבודה (אלגוריתם) ויישמנו אותה בתוך המחלקה היחידה שבתוכנית (ניקרא לה המחלקה הראשית), ובתוכה, בתוך הפעולה שנקראה Main

עכשיו נגלה שבגלל ש C# היא שפה מונחית עצמים, כל תוכנית מורכבת לפחות מחלקה אחת. את שמה אנחנו יכולים לקבוע כרצוננו (סבבת העבודה הנותנת כבר שם קבוע מראש, אבל אפשר לשנותו).

התחלנו לראות שאפשר גם להגדיר טיפוסים (מחלקות) משלנו בשפה, כמו המחלקה Circle שהגדרנו בשיעור הקודם.

מה היה המבנה של מחלקה חדשה:

```
public class name
{
    //Properties (variables)
    //Constructor (could be more than 1)
    //Methods
}
```

התכונות או המאפיינים של המחלקה (טיפוס) אמרנו שהם בד"כ מוגדרים פרטיים (private)

כל עצם שניצור בעזרת המחלקה החדשה תהיינה לו התכונות שהגדרנו במחלקה.

## הבנאי (או הפעולה הבונה, או הבונה או ה- constructor)

- (1) אם יש צורך, הוא מקצה שטחים בזיכרון (בינתיים לא היה צורך בדוגמה של Circle)
- (2) הוא מאתחל עם ברירת המחדל את המאפיינים על פי ברירות המחדל שלהם, גם אם אין לו שום פעולה ביצועית. לדוגמה:

```
using System;

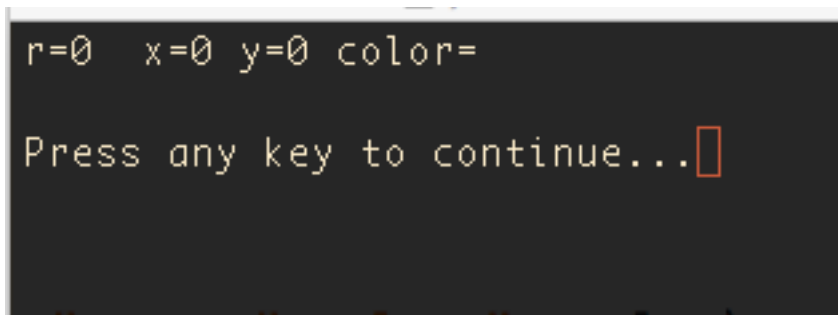
namespace loops
{
    class MainClass
    {
        public class Circle
        {
            private int r,x,y;
            private string color;

            public Circle(int r_init) //empty constructor
            {
            }

            public void Printob()
            {
                Console.WriteLine("r="+r+" x=" +x + " y=" +y
                    + " color=", color);
            }
        }

        public static void Main(string[] args)
        {
            Circle c1 = new Circle(10);
            c1.Printob();
        }
    }
}
```

כך נראה הפלט:



```
r=0 x=0 y=0 color=
Press any key to continue...
```

כלומר המשתנים (תכונות) השלמים קיבלו אפס והמחרוזת (צבע) קיבלה רווחים ריקים.

המשך הבנאי:

3) גוף בנאי מתבצע (שם אפשר לכתוב איזה פקודות שצריך), הבנאי של Circle בסך כל עשה השמה של ערכי הפרמטרים לתכונות

4) הבנאי 'מחזיר' (או יוצר) עצם חזרה למרות שאין שום פקודה מפורשת שאומרת לו להחזיר משהו.

זהו המכניזם של הבנאי.

### תרגיל כיתה

כתוב בנאי נוסף למחלקה Circle שתפקידו יהיה רק להכניס ערך לצבע (הוסף כמובן את תכונת הצבע למחלקה).