

# שני מערכי שיעור (2 \* 90 דק') :

## א. "היכרות עם סביבת העבודה ועצמים"

## ב. "הוראת משפט IF פשוט"

### 1. כללי

---

#### א. סביבת עבודה

- נעבוד בסביבה של Window Application. סביבה המכילה בקירבה עצמים הניתנים לשימוש מיידי, תוך כדי גרירה בלבד, וזאת בלי דרישה של ידע מוקדם.
- סביבה זו מאפשרת לנו להבין בקלות את המהות של עבודה עם עצמים.
- בסביבה זו ניתן לבנות בקלות תוכניות (משחקים) שיגרמו לתלמידים לאהוב את עולם התיכנות ולהתקרב אליו.
- הכוונה לעורר בתלמידים אלו את הרצון להמשיך ולבחור במגמה זו לאחר מכן, אי לכך חשוב שהשיעור יועבר בצורה מעניינת ומגרה אך לא בצורה קשה ומורכבת מדי ....

#### ב. נושאי השיעורים

- הכרה עם סביבת העבודה WindowApplication ועם שיט עבודה EventDriven ( מונחה ארועים).
- פתיחה לנושא הוראת התנאי, הצורך בה, שימושיה מבנה ההוראה הפשוטה והגדרתה. הדגש יושם על העובדה שלא כל קטעי האלגוריתם מבוצעים בכל פעם.

#### ג. ידע מוקדם

- אין צורך, השיעורים מוגדרים כשיעורים ראשוניים בסביבת העבודה.

## ד. מטרת השעור

- התלמיד ילמד לעבוד בסביבת window Application
- התלמיד ילמד לעבוד בסביבת EvenDriven, תכנות מונחה אירועים.
- התלמיד ילמד לעבוד עם עצמים, כמו כפתור, ועם התכונות שלו (כגון צבע).
- התלמיד יכיר בצורך בשימוש בהוראת תנאי.
- התלמיד ינסח הוראה לביצוע תנאי באלגוריתם מילולי.
- התלמיד יכיר את מבנה הוראת תנאי פשוט ב C#.
- התלמיד יפתור בעיות תוך שימוש בהוראת התנאי הפשוט.
- התלמיד יכיר בצורך בשימוש במשתנה
- התלמיד ילמד לעבוד עם משתנה int
- התלמיד יכיר בייחודיות של משתנה bool
- התלמיד ילמד לעבוד עם משתנה bool

## 2. מהלך השיעורים:

### א. מבנה השיעורים:

#### שיעור היכרות

- השיעור בכללותו הוא באורך שתי שעות הוראה פרונטליות (2 \* 45 דקות).
- **הצגת הסביבה:** (זמן משוער 15 דקות)
- **הצגת משימת היכרות, והשלב הכרוכים בה:** (זמן משוער 20 דקות)
- **תרגול עצמי:** (זמן משוער 50 דקות)
- **סיכום:** (זמן משוער 5 דקות)

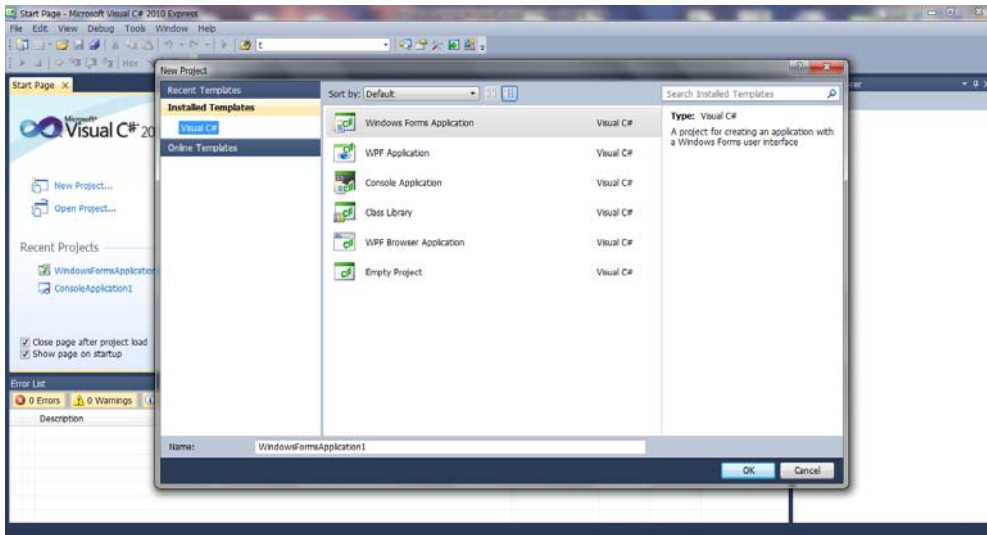
#### שיעור לימוד ההוראה IF והיכרות עם שימוש במשתנים (מסוג int ו-bool)

- משימת חזרה על השעור הקודם (רמזור): (זמן משוער 15 דקות)
- הצגת מבנה המשפט של הפקודה If והשימוש בו: (זמן משוער 15 דקות)
- סיום משימת הרמזור, כולל התנאי: (זמן משוער 10 דקות)
- הצגת הצורך והשימוש במשתנה מסוג int: (זמן משוער 10 דקות)
- הצגת הצורך והשימוש במשתנה מסוג bool: (זמן משוער 10 דקות)
- תרגול עצמי: (זמן משוער 30 דקות)

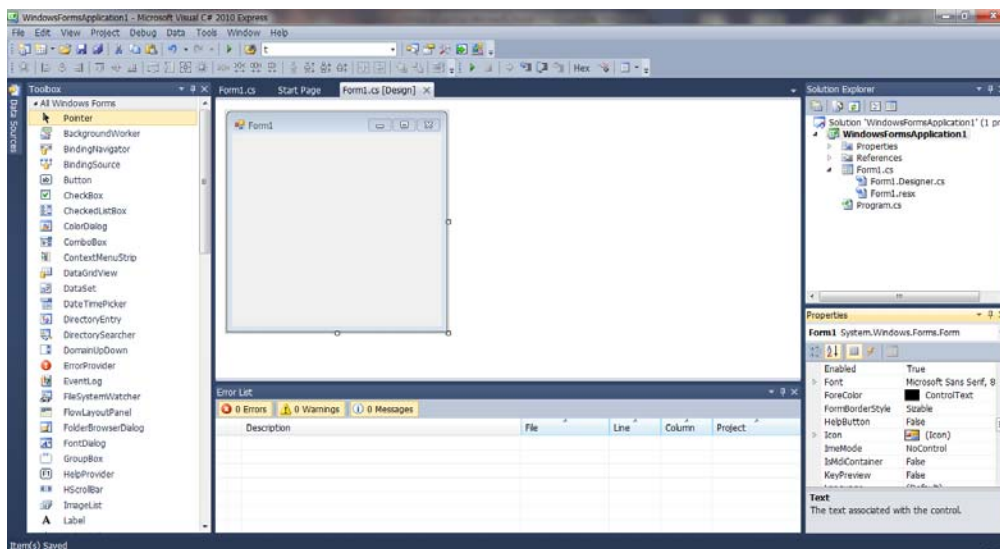
## ב. היכרות עם הסביבה

יצירת תוכנית / פרויקט


על מנת ליצור תוכנית חדשה נלחץ על הלחצן New Project → File, כך שייפתח המסך הבא:

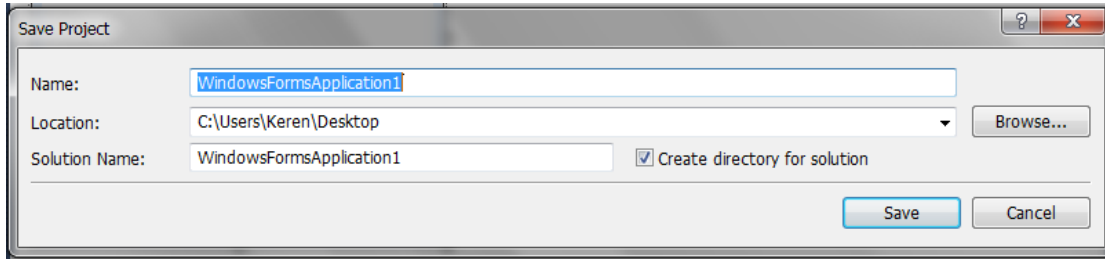


מוצגות לפנינו מספר אפשרויות לפרויקטים שונים מביניהם אנו נבחר בסביבת העבודה WindowsFormsApplication. נלחץ פעמיים ויופיע לנו המסך הבא:




## שמירת תוכנית / פרויקט

על מנת לשמור פרויקט חדש בדרך כזו שנוכל לשנותו ולהשתמש בו בעתיד נלחץ על סימן השמירה הבא:  שמופיע בשורת המשימות בראש העמוד. לאחר הלחיצה על הסמל יופיע לנו החלון הבא:



בשורה Name נכתוב את שם הפרויקט שלנו; ובשורה Location נבחר היכן נשמור את הפרויקט במחשב. הפרויקט יישמר באופן אוטומטי בתוך תיקייה ששמה הוא כשם הפרויקט (אלא אם ביטלנו את סימון התיבה "Create directory for solution").

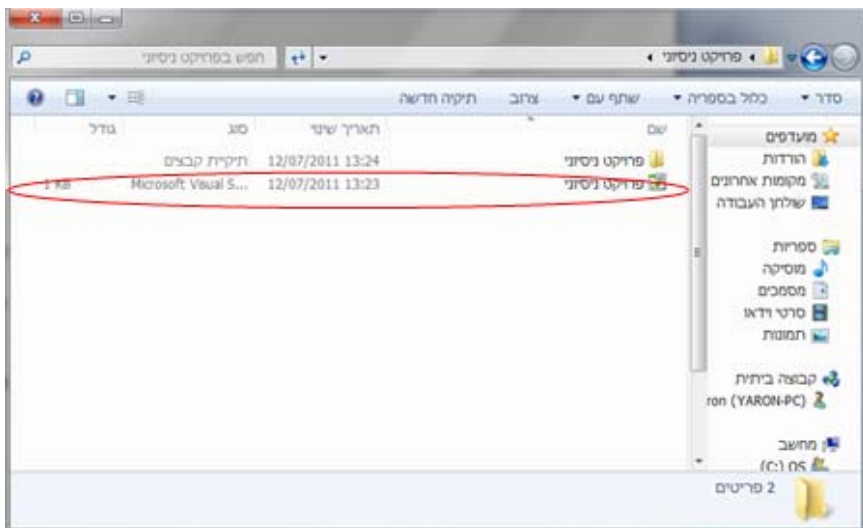
מרגע ששמרנו תוכנית פעם אחד באופן זה, בשמירה הבאה לא נצטרך לציין מחדש את שם הפרויקט ואת המיקום שבו נרצה לשמור אותו. רק נצטרך ללחוץ על הסמל , וכך יישמרו השינויים שעשינו בתוכנית.

## פתיחת תוכנית / פרויקט

נדגים באמצעות פרויקט אשר נקרא "פרויקט ניסיוני", על מנת לפתוח את הפרויקט הנוכחי במחשב נלחץ פעמיים כדי לפתוח את התיקייה שבה שמרנו את



הפרויקט: לאחר מכן נלחץ על הפרויקט עצמו כפי שמתואר במסך הבא:



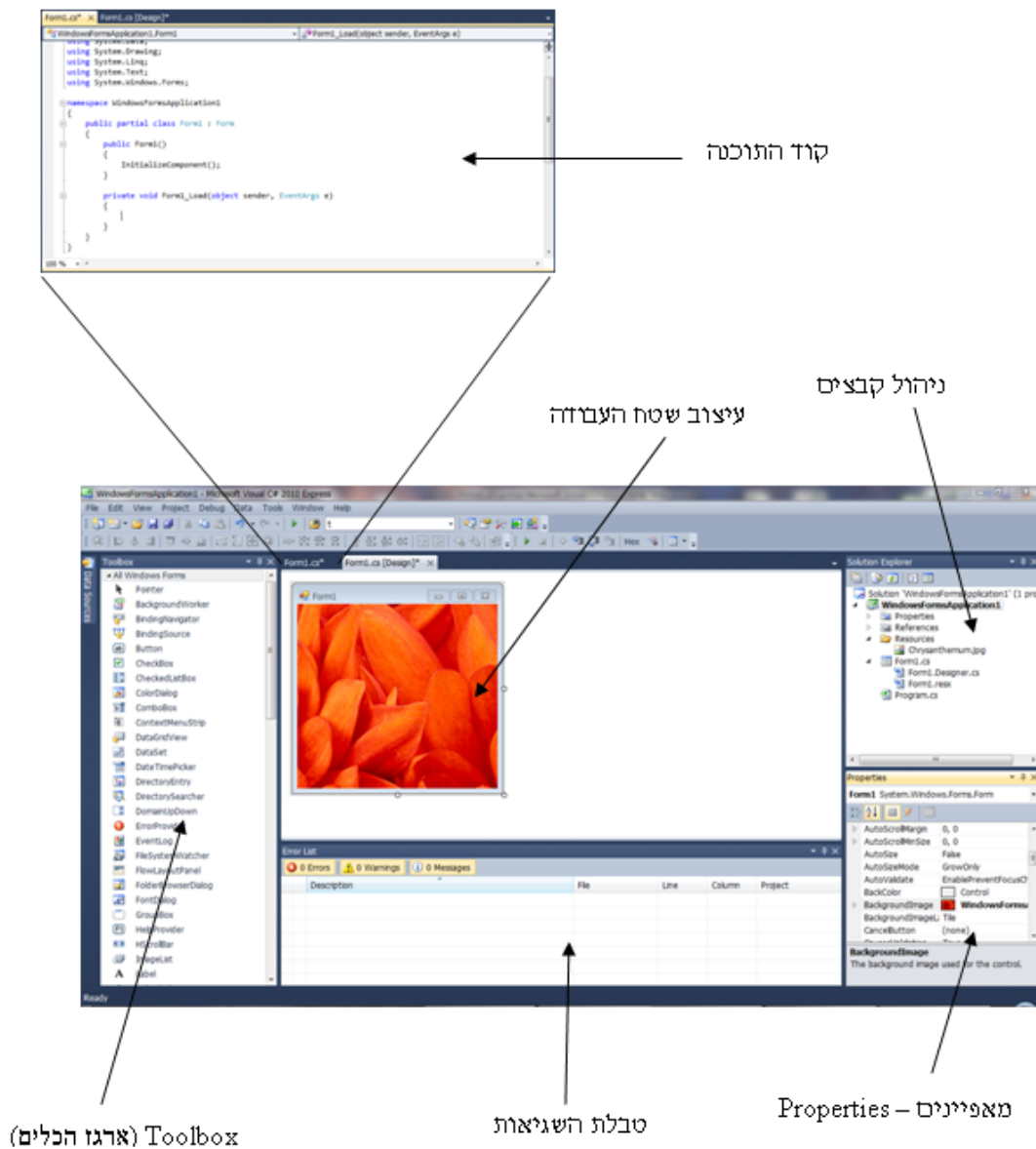
שם	תאריך שינוי	סוג	גודל
פרויקט ניסיוני	12/07/2011 13:24	תיקיית קבצים	
Microsoft Visual S...	12/07/2011 13:23	קובץ	1 כ"ב

במצב זה יפתח לנו מסמך התוכנה ונוכל לבצע בפרויקט שינויים כרצוננו.

## מבנה הסביבה

סביבת ה- WindowsFormsApplication מורכבת משישה חלקים איתם נעבוד והם:

1. Toolbox (ארגו הכלים)
2. עיצוב שטח העבודה
3. ניהול קבצים
4. קוד התוכנה והרצת הפרויקט
5. מאפיינים - Properties
6. טבלת השגיאות

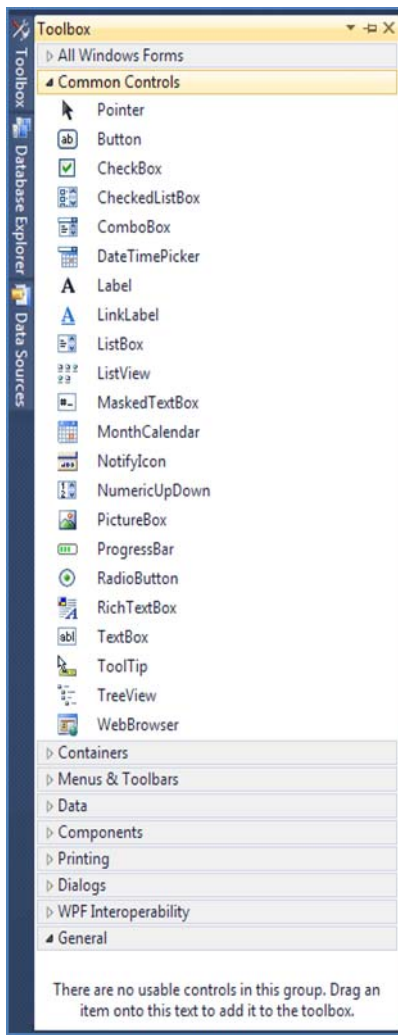


## Toolbox (ארגז הכלים)

ה- Toolbox (ארגז הכלים) מאחסן את כל הפריטים בהם נשתמש לתוכניות שלנו. כל פריט שנרצה להוסיף לתוכנית (למשל כפתור - Button, תיבת טקסט - TextBox וכו') - נגרור אותו עד לטופס שלנו (ל-Form), ונמקם אותו כרצוננו.

במקרה שאנו לא רואים את ה- Toolbox במסך, כלומר הוא מוחבא, יש ללחוץ על View בתפריט העליון, ואז ללחוץ על Toolbox.

ב- Toolbox, בפינה הימנית-עליונה, יש חץ קטן (▼). כשנלחץ עליו נראה כמה מצבים:



1. Floating: במצב זה ניתן לגרור את ה- Toolbox כולו ממקום למקום - הוא "צף" על המסך. מצב זה אינו מומלץ משום שזה יוצר אי-סדר.

2. Dockable: ה- Toolbox יהיה בצד שמאל של המסך; זוהי השיטה המומלצת ביותר מפני שכך תהיה לנו גישה ישירה אליו בזמן עיצוב התוכניות שלנו.

3. Tabbed Document: ה- Toolbox יתפוס מסך שלם. בדרך כלל לא נשתמש בשיטה זאת מפני שממילא אנו משתמשים בפרויקט שלנו עם דפי תכנות רבים, ולכן ולא נרצה שיתווסף לנו דף נוסף.

4. Hide: ה- Toolbox יוחבא, ושוב נצטרך ללחוץ על View על מנת לראות אותו.

5. Auto Hide: אפשר יהיה לראות את ה- Toolbox רק אם נהיה עם העכבר מעל הכיתוב של ה- Toolbox בצד שמאל. מצד ימין של החץ ▼ יש כפתור של נעץ. כאשר הוא אופקי, ה- Toolbox יהיה במצב של Auto Hide; כאשר הנעץ אנכי ה- Toolbox יהיה במצב של Dockable. בשיעורים שלנו נגדיר תמיד את ה- Toolbox למצב של Dockable.

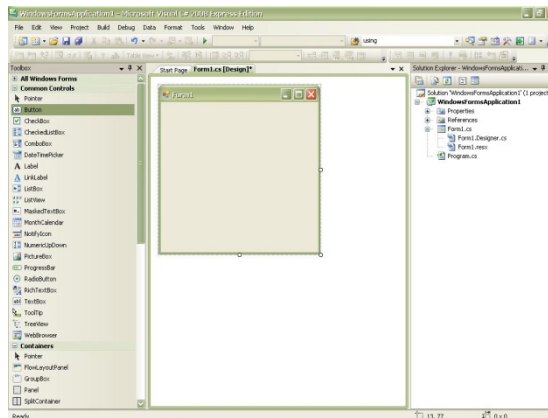
בתוך ה- Toolbox קיימות כמה קטגוריות לתצוגת הפריטים. השתיים החשובות לנו הן:

1. All Windows Forms - מציג את כל הפריטים הקיימים.

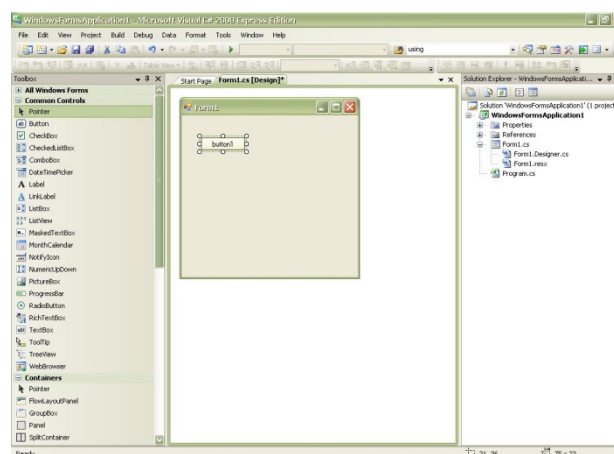
2. Common Controls - מציג את הפריטים הנפוצים (אנו נשתמש לרוב בפריטים שתחת הקטגוריה הזו).

כעת, ניתן דוגמה של גרירת פריט מה- Toolbox אל הטופס שלנו (אל ה- Form). נבחר פריט הנקרא Button, ונגרור אותו לטופס.

1. נבחר פריט לדוגמה - Button :



2. נגרור את ה-Button אל הטופס ונמקם אותו במקום שנרצה. (אפשר לשנות את הגודל של הפריט ואת מיקומו גם אחרי שמיקמנו אותו בטופס):



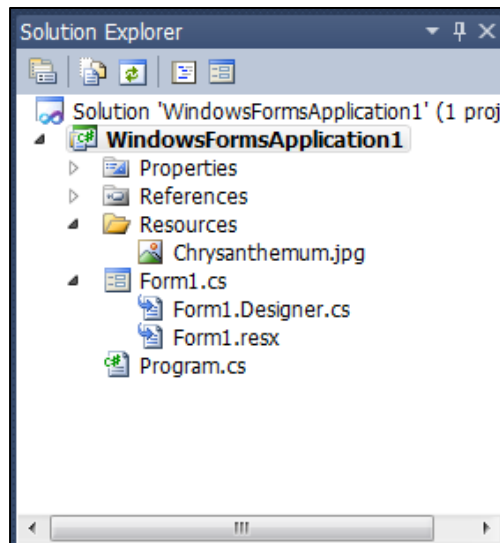
## עיצוב שטח העבודה

עיצוב שטח העבודה הוא למעשה המקום בו נעצב כיצד יראה מסך הטופס שלנו (ה- Form). הטופס מכיל את הפריטים שגררנו אליו מה- Toolbox, ו"עליו" פועלת התוכנית:



### 3. ניהול קבצים בפרויקט

הפרויקט עצמו אינו בנוי מקובץ אחד אלא ממספר קבצים אשר יוצרים תוכנית אחת. בסביבת העבודה שלנו ניהול הקבצים והמעבר ביניהם יהיה בעזרת החלון העליון הימני ביותר:



בעזרת חלון זה נראה אילו קבצים פועלים ונוכל לעבור ביניהם ולשנות אותם לפי ראות עינינו.

### קוד התוכנה והרצת הפרויקט

בחלקים הקודמים של מבנה הסביבה התייחסנו בעיקר אל העיצוב של פרויקט התוכנה שלנו. אולם בתכנות קיים מאפיין נוסף: כתיבת הקוד עצמו. אנחנו נכתוב את הקוד בשפת התכנות שלנו (C#) כדי להשיג מטרה מסוימת.

כפי שהוצג לעיל, אחד מהחלקים של סביבת העבודה הוא קוד התכנות. הקוד מופיע כאשר אנו לוחצים על טופס העבודה שלנו פעמיים או כאשר אנו לוחצים על כל עצם אשר מופיע בטופס פעמיים.

דרך נוספת להגעה לחלק שבו כותבים את הקוד הוא לחיצה על המקש F7.

לאחר הפעם הראשונה בה הגענו אל הקוד, תופיע כרטיסיה לכך כפי שראינו בתרשים שבעמוד 22. כך נוכל גם לעבור לקוד על ידי לחיצה פעמיים בעכבר, על-ידי לחיצה על הכרטיסייה המתאימה

בכרטיסיית הקבצים, או על-ידי לחיצה על המקש F7.

קוד התוכנה נראה כך:

```
Form1.cs* x Form1.cs [Design]*
WindowsFormsApplication1.Form1
using System;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            |
        }
    }
}
```

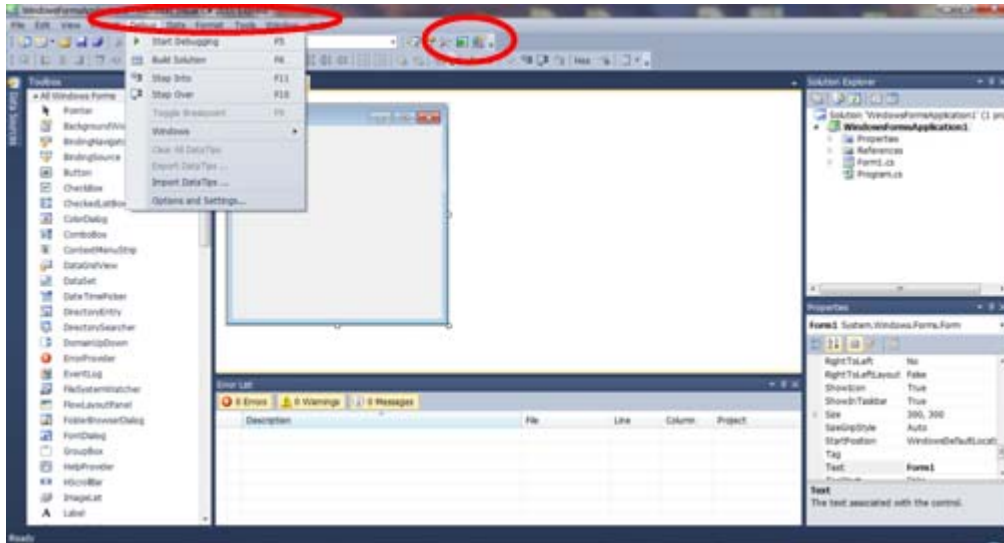


## כיצד נריץ את התוכנית שלנו?

בשביל להריץ את התוכנית נשתמש בכפתור Debug שמשמעותו ניפוי-שגיאות בתוכנית. למעשה הדבר בודק האם יש שגיאות בתוכנית: אם כן, הוא מזהיר מפניהן ומראה לנו אותן בטבלת השגיאות; אם לא, הוא מריץ את התוכנית ומבצע אותה על-פי הקוד שכתבנו.

נריץ את התוכנית על-ידי שימוש ב-Debug בשלוש דרכים שונות:

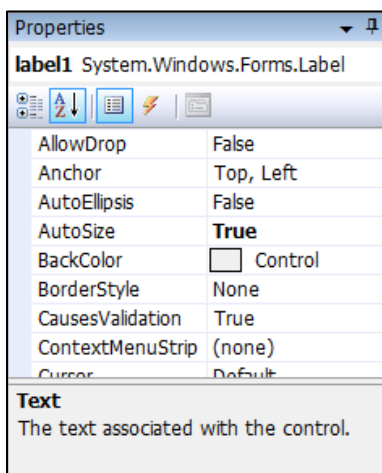
1. על-ידי לחיצה על המשולש הירוק בסרגל הכלים למעלה
2. על-ידי לחיצה על **Debug** ← **Start Debugging**
3. על-ידי לחיצה על **F5**



הערה: ניתן להריץ תוכנית גם ללא ניפוי-שגיאות. עושים זאת בשתי דרכים שונות:

1. על-ידי לחיצה על **Debug** ← **Start Without Debugging**
2. על-ידי לחיצה על **Ctrl + F5**.

אף על פי שאפשרות זו קיימת, לא נמליץ להשתמש בה משום שכך לא נקבל התראה על שגיאות בתוכנית.



## מאפיינים - Properties

בתוכנית שלנו מופיעה טבלת מאפיינים שמצהירה על מאפיינים שונים של עצמים בכלל, ועצמים שגררנו מה- Toolbox בפרט. כך נוכל לשנות מאפיינים כגון צבע, טקסט, גודל וכו'. טבלת המאפיינים בתוכנה שלנו נמצאת בצד ימין למטה והיא נראית כך:

בעזרת טבלה זאת נוכל לשנות את המאפיינים של העצמים. כדי לשנות מאפיינים של עצם מסוים, נלחץ עליו פעם אחת. כתוצאה מכך מאפייניו יופיעו בטבלה.



## הערות בשפת C#

בשפת C# נוכל להוסיף הערות. הערות אלה משמשות את קוראי קוד התוכנית בלבד, על כן הן עוזרות לא רק למתכנת התוכנית להבין את התוכנית שלו ביתר קלות על ידי הצבת הערות שיזכירו לו מה כל חלק בתוכנה עושה, אלא הן עוזרות למתכנתים אחרים להבין למה התכוון מתכנת התוכנית. כך תוכניות אשר יועברו ממתכנת אחד לשני או ייפתחו אחרי זמן רב יהיו מובנות בקלות רבה. בזמן הרצת התוכנית המהדר מתעלם מההערות וממשיך לפקודה הבאה. על ידי כתיבת ההערות נוכל להבין יותר טוב את הקוד, להיזכר בשינויים שעשינו ולגלות מקור בעיות במהירות רבה יותר.

### את ההערות ניתן לכתוב בשתי צורות עיקריות:

צורת הכתיבה הראשונה: הערה המתפרסת על שורה אחת.

```
!
// → פה תיכתב הערה
!
```

את ההערה הזאת אנו כותבים על ידי צמד לוכסנים // ואחריהם נכתוב את ההערה שרצינו. בשפת C# ההערה תיבדל משאר הקוד בכך שתיצבע בירוק.

### שימו לב!



סוג זה של ההערה מחזיק אך ורק בשורה שבה נכתבה ההערה. על מנת לכתוב הערה שתתפרס על כמה שורות יש להשתמש בצורת הכתיבה השנייה.

צורת הכתיבה השנייה: הערה המתפרסת על כמה שורות.

```
!
/* תחילת ההערה
!
!
!
*/ סוף ההערה
!
```

את ההערה הזאת פותחים על ידי הסימן /\* (לכסן ולאחריו כוכבית), וכשמסיימים לכתוב את ההערה סוגרים אותה על ידי הסימן \*/ (כוכבית ולאחריה לוכסן). גם בצורת הכתיבה הזאת ניתן לכתוב הערות המתפרסות על שורה בודדת.

### דוגמה להערות:

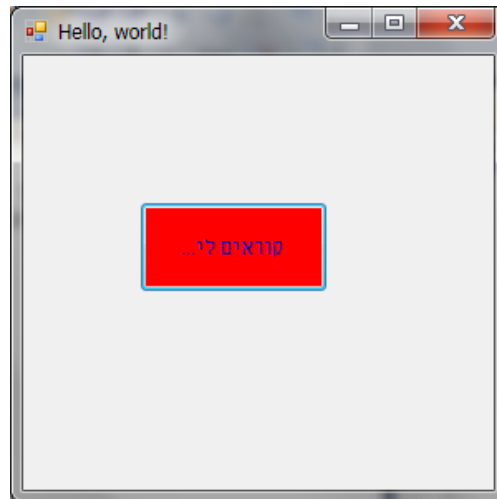
```
public partial class Form1 : Form //this is where we program
{
    /*here you write the program...
    * ...this is still comment*/
}
```

## ג. משימת היכרות :

משימת ההיכרות תהיה משימת התכנות הראשונה שלנו, **אותה נפתור יחדיו** עם התלמידים. בעזרת משימה זו נלמד את יסודות התכנות הבסיסיים ביותר אשר יבואו לידי ביטוי במהלך כל השנה.

בתור המשימה הראשונה שלנו, נכתוב בה **"Hello, world!"** כנהוג בקרב מתכנתים בכל העולם.

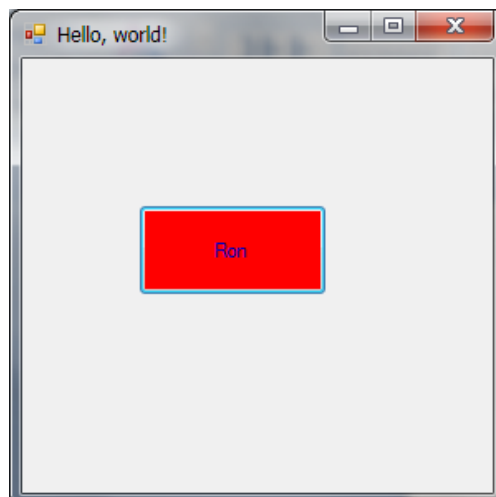
טופס המשימה שלנו ייראה בתחילה כך :



המשימה שלנו תהיה זאת :

1. נשנה את כותרת הטופס ל-"Hello, world!".
2. ניצור פריט הנקרא כפתור (**Button**).
3. על הכפתור יהיה כתוב: "קוראים לי...".
4. העיצוב בכפתור יהיה: צבע הטקסט- כחול, רקע הכפתור- אדום.
5. לאחר שנלחץ על הכפתור, שם התלמיד שכתב את התוכנית יופיע בכפתור.

אחרי שנלחץ על הכפתור, כלומר לאחר שביצענו את המשימה, יופיע, לדוגמה :



## כיצד נבצע את כל הדברים הללו?

הנושאים שנלמדים בפרק זה הם:

2. כפתורים – Buttons
3. המאפיין Name – שם של עצם
4. פנייה לעצמים בקוד התוכנית ושינוי מאפייניהם
5. מחלקת Color
6. תמונות (לא נחוץ לפתרון הבעיה)
7. פעולות של עצמים
8. תוויות - Labels (לא נחוץ לפתרון הבעיה)

## כפתורים - Buttons

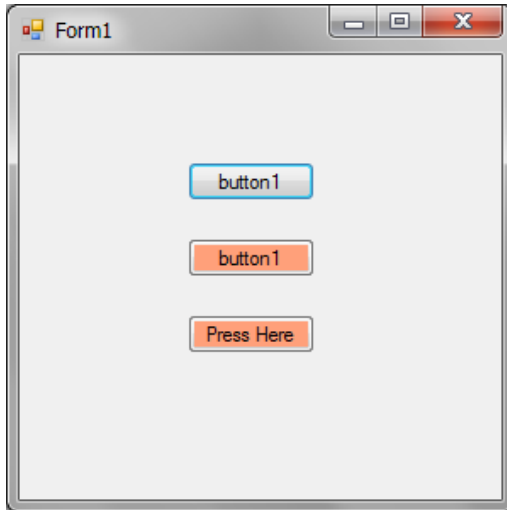
כפתור (Button) הוא אחד הפריטים הכי פופולרים ופשוטים הקיימים ב-Toolbox. מטרת הכפתור היא להפעיל פעולה מסויימת (קוד מסויים) כתגובה לחיצה על הכפתור. לדוגמה, במשימת ההיכרות שלנו לחיצה על הכפתור תגרום להופעת שם התלמיד.

לכל פריט (במקרה שלנו **Button**) קיימים מאפיינים אותם ניתן להגדיר עוד לפני הפעלת תוכנית המחשב (במצב עיצוב design). המאפיינים מוגדרים בחלון שנקרא **Properties**. בכל פעם מוצגים בחלון המאפיינים רק אותם המאפיינים אשר שייכים לעצם המסוים בטופס שסומן כנבחר. לכל עצם מאפיינים רבים, ואנו נלמד להפעיל ולשלוט על מספר מהמאפיינים היותר כלליים ופופולאריים (תלמידים טובים יוכלו להבין בעצמם ולהגיע ליותר מאפיינים) נזכור כי אנו יכולים לצרף אל התוכנית שלנו כמות כפתורים כרצוננו אך לכל כפתור ישנו שם שונה אשר דרכו הוא מזהה ע"י המחשב (**Name**).

נגרור את ה- **Button** לטופס, נלחץ עליו ונראה את חלון ה- **Properties**.

כעת נציג את המאפיינים של ה- **Button** בהם נשתמש:

1. **BackColor** - צבע הרקע של הכפתור.
2. **Text** - הכיתוב (הטקסט) על הכפתור.
3. **Font** - אפשר לבחור את גודל הטקסט, אם רוצים שיהיה בולט (**Bold**), אם רוצים שיהיה לו קו תחתון (**Underline**) וכו'.
4. **ForeColor** - צבע הטקסט שבכפתור.
5. **Visible** - מאפיין שמגדיר האם לאפשר למשתמש לראות את הכפתור או לא. למאפיין זה שני ערכים אפשריים: **True** או **False**. בחירה ב- **True** תאפשר לראות את הכפתור; בחירה ב- **False** תגרום לכפתור להיות בלתי נראה (אף על פי שהוא עדיין קיים).
6. **Enabled** - הפעולה הזאת גורמת לכך שהכפתור לא יהיה פעיל כלומר הוא לא יהיה "בר לחיצה".



לדוגמה נוכל לראות את התוכנית הבאה :

לפנינו שלושה כפתורים. הכפתור הראשון הינו כפתור רגיל וזוהי ברירת המחדל שלו. הוספנו עוד שני כפתורים ושינינו את מאפיינים בטבלת המאפיינים. ניתן לראות כי שינינו את רקע הכפתור הראשון ואת רקע הכפתור השני. בנוסף, שינינו את הטקסט שבכפתור השלישי.

## ◀ המאפיין Name - שם של עצם

כפי שהסברנו עד עכשיו, לכל עצם בפרויקט שלנו ישנם מאפיינים שונים אשר מופיעים בטבלת המאפיינים.

עד עתה נתקלנו במאפיינים רבים של הכפתור כמו שינוי צבע רקע הכפתור, שינוי Font והיעלמות הכפתור ממסך התוכנית שלנו.

נכיר עכשיו מאפיין חדש שלא תקף רק לכפתורים, אלא לכל העצמים אשר מצויים בסביבת העבודה שלנו - המאפיין Name.

המאפיין Name מופיע בראש טבלת המאפיינים. לפי שמו, אנו מסיקים שהוא אחראי לשם של העצם שלנו. הכוונה היא לשם שבאמצעותו נוכל להתייחס לעצם בכתיבת הקוד.

הערך ההתחלתי שלו, ברירת המחדל, מופיע עם הגדרת העצם.

לדוגמה, הכפתור הראשון שיצורף לתוכנית ייקרא בשם **button1**.

אולם נזדקק להבנה כי קביעת השם (לשנות אותו מברירת המחדל) מאוד חשובה בעת יצירת פרויקט והיא מייעלת לנו את התוכנית מאד.

שינוי השם מאפשר לנו להבדיל בקלות בין העצמים השונים. בנוסף לכך, התוכניות נעשות קריאות ונוחות הרבה יותר.

תארו לעצמכם מה היה קורה אם הייתה לנו תוכנית בעלת 20 כפתורים שונים, כך שלכל כפתור תפקיד שונה. לא היינו זוכרים איזה כפתור מבצע איזה תפקיד, היינו מסתבכים בכתיבת התוכנית, והיה לוקח זמן רב לסיים אותה בהצלחה שכן סביר שיהיו לנו שגיאות אלגוריתם רבות.

**נציג עתה מספר כללים אשר יאפשרו לנו לבחור שם בצורה היעילה ביותר לתוכנית הספציפית שלנו:**

1. השם יהיה באותיות קטנות, אלא אם מתחילה באמצעו מילה חדשה. במקרה כזה, האות הראשונה של המילה הזו תהיה אות גדולה.

2. ניתן לעצם שם הגיוני שמתאר את המטרה שלשמה הוא צורף לתוכנית.

3. בסוף השם נוסיף את הסוג של העצם (למשל, עבור כפתור נוסיף את המילה **button**).

4. נהוג לקצר מילים כדי שהשמות לא יהיו ארוכים מדי. עם זאת אין להגזים ולעבור את גבול הטעם הטוב.

לדוגמה: עבור כפתור שאחראי לשינוי צבע הרקע של הטופס נשתמש בשם:

**formBackColorButton** ניתן לקצר את השם ל- **formBClrBtn**.

## שימו לב:



יש להקפיד ולשנות את המאפיין **Name** של העצמים בתוכניות שלכם, למען הבהירות והקריאות.

## פנייה לעצמים בקוד התוכנית ושינוי מאפייניהם

יש לדעת כי נוכל לשנות את המאפיינים של עצמים לא רק על ידי דרך טבלת המאפיינים, אלא גם לכתוב דרך הקוד. בעמוד זה ניתן דוגמה לקוד על כפתורים; אך מובן שהדבר יהווה מבנה כללי לשאר העצמים שנלמד השנה.

על מנת לפנות אל עצם ולשנות את מאפייניו נשתמש במבנה הבא:  
ניסוח הפקודה בעברית:

```
ערך = מאפיין.שם_העצם.this;
```

משמעות השורה היא שאנו פונים אל העצם (שם\_העצם – שהוא ערך המאפיין **Name** של העצם) ששייך לטופס הנוכחי (**this**) ומבקשים לשים במאפיין ערך.

אין חובה לכתוב את המילה **this** בתחילת השם, אבל...ככה זה נכון

ניתן דוגמה על ידי שימוש בכפתורים אשר נלמדו:

על מנת לבצע את שינוי הטקסט בדוגמה שהצגנו בעמוד 31 דרך הקוד, נעשה את הדבר הבא:

```
this.button3.Text = "Press Here";
```

נסביר את השורה:

פנינו אל כפתור מס' 3 שמזוהה בקוד על-ידי השם **button3** (תמיד נוכל לשנות את השם, ורצוי לעשות זאת, על ידי שינוי המאפיין **Name**; נשנה את השם לשם משמעותי בטופס), וביקשנו שערך המאפיין **Text** שלו יהיה "Press Here".

כמו כן אנו רואים כי בסוף השורה נמצא הסימן ; (נקודה-פסיק).  
מכאן נזכור:

## בל הוראה שנכתוב בשפת C# תסתיים בנקודה-פסיק (;).

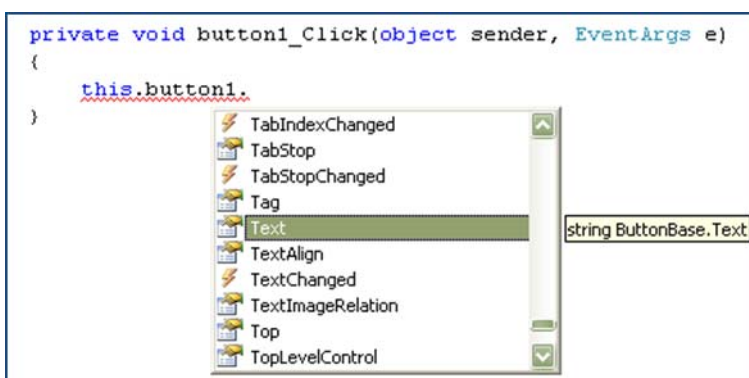


מכך נסיק:

כאשר נרצה להתייחס ל- **Button** מסוים בקוד, בשם **button1**, לשם שנוי מאפיינים למשל, נכתוב **this.button1**. אחרי שנכתוב את הנקודה שלאחר שם הכפתור (**button1** במקרה זה), תופיע רשימה עם (בין היתר) כל המאפיינים של הכפתור לפי סדר האלפבית באנגלית, מתוכה נבחר את המאפיין שאותו אנו רוצים להגדיר או לשנות.  
ובאופן כללי כל פעם שנכתוב:

```
שם_עצם.this.
```

נוכל לראות רשימה של מאפיינים (ועוד) שבהם נשתמש.

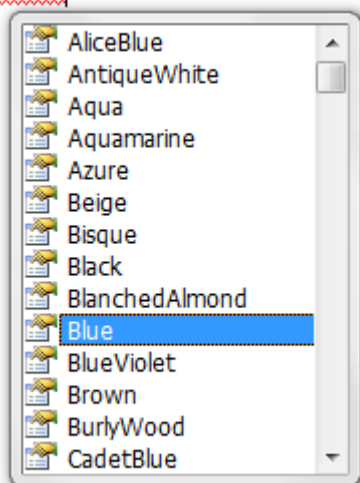


כמו כן בדוגמה שלנו ראינו כי ניתן לשנות את רקע הכפתור.  
בשביל לדעת איך לעשות זאת נלמד על מחלקת **Color**.

## מחלקת Color

מחלקת **Color** היא מחלקה אשר נוצרה על ידי כותבי השפה לשימוש המתכנתים בה.  
מחלקה זאת, לפי שמה, נותנת לנו גישה לצבעים שונים ומגוונים מחוץ לטבלת המאפיינים שלנו.  
על מנת לפנות אל צבע מסוים בקוד התוכנה שלנו, נכתוב את הקוד הבא:

Color.



כאשר נכתוב מילה זאת יפתחו בפנינו מגוון צבעים בהם  
נוכל לעשות שימוש.

אך איך למעשה צבענו את רקע הכפתור?  
כתבנו את שורת הקוד הבאה:

```
this.button2.BackColor = Color.Salmon;
```

מה שעשינו בשורת קוד זאת הוא לפנות אל הכפתור  
הרצוי, ללכת אל מאפיין צבע הרקע שלו ולשנות את ערכו  
על ידי שימוש במחלקת **Color**.

## תמונות (מיועד לתלמידים יותר מתקדמים)

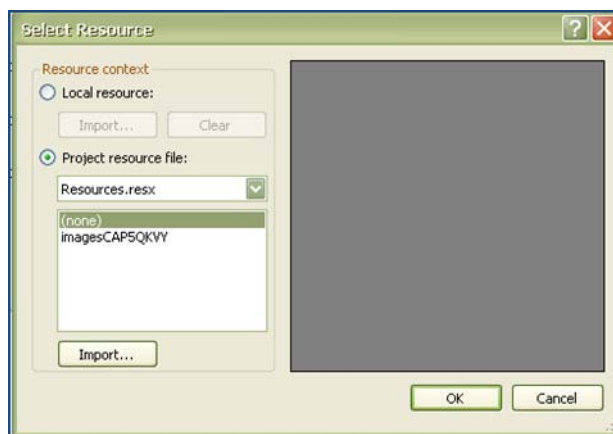
על מנת לייפות את התוכנית שלנו ולעשות אותה יותר יפה ויזואלית נשתמש בתמונות אשר נייבא  
ממקומות שונים במחשב, מהאינטרנט וכו'.

נוכל לעשות שימוש רב בתמונות:

1. נוכל להשתמש בתמונות בתור ערכים של מאפייני עצמים. למשל ניתן לשים תמונה כרקע של  
כפתור, שהוא עצם. לשם כך נצטרך לייבא את התמונה הרצויה.  
נפעל על-פי הצעדים הבאים (דוגמה להשמת תמונה כרקע של כפתור):



נלך לטבלת המאפיינים ונלחץ על:  
יופיע לנו החלון הבא:





יש ללחוץ על **Project resource file**, ואז ללחוץ על **Import** על מנת לייבא את התמונה לרקע של הכפתור. לאחר ייבוא התמונה נסיים את התהליך על-ידי לחיצה על הכפתור "OK".

**שימו לב**- על מנת לראות את התמונה בצורה ברורה על הכפתור, יש להשתמש במאפיין **BackgroundImageLayout** (ראו בהמשך העמוד).



דוגמה לכפתור עם תמונת רקע:

### המאפיין BackgroundImageLayout

מאפיין זה מייצג את האופן שבו תמונת הרקע של הכפתור תיראה. למאפיין זה יש מספר ערכים אפשריים, ועלינו לבחור אחד מהם. נציג שניים מהם:



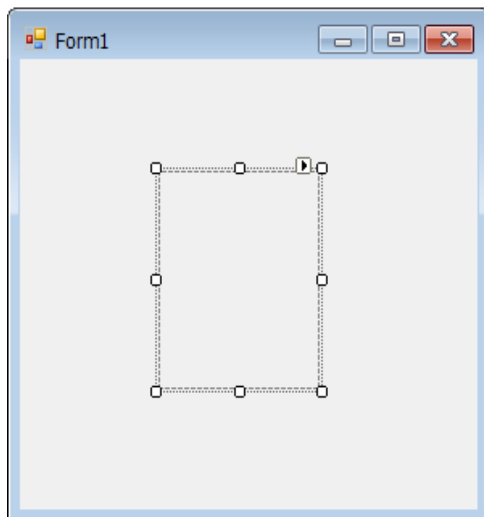
א. **Stretch**- התמונה מתאימה את עצמה לגודל הכפתור (נמתחת):



ב. **Zoom**- התמונה תהיה יותר קטנה מהכפתור עצמו:

2. נוכל לייצר תמונות אשר יופיעו לנו סתם כך לקישוט הטופס שלנו. נשתמש ברכיב שנקרא: **PictureBox**. ה**PictureBox** הוא עצם לכל דבר, בדיוק כמו הכפתורים, ועל כן נוכל לייצר עצמים רבים מהסוג **PictureBox**, והם ייבדלו על ידי שמות שונים במחשב (נזכיר שרצוי לשנות את השמות באמצעות המאפיין **Name**).

כאשר נצרף את **PictureBox** מסוים אל הטופס שלנו הוא ייראה כך:



כמו בכפתורים, בסעיף הקודם, ניכנס אל המאפיין **Backgroundimage**, ונייבא תמונה. גם כאן נשתמש במאפיין **BackgroundImageLayout**.

## מחיקת תמונה

כאשר נרצה למחוק תמונה יש למחוק את הכתובת שלה מהמאפיין `BackgroundImage`:



## פעולות על עצמים

עתה, לאחר שלמדנו על המאפיינים של הכפתור ושל התמונה, נלמד קצת על הקוד שמאחוריהם ואיך למעשה נבצע את משימת ההיכרות שלנו. אולם נושא זה לא מכסה ר תמונה וכפתור אלא מעתה כל פעם שנרצה לבצע פקודות על עצמים נבצע אותן בתוך הפעולות של העצמים.  
כאשר נלחץ פעמיים על כפתור (שה- `Name` שלו הוא `button1`), תופיע לנו הפעולה:

```
private void button1_Click(object sender, EventArgs e)
{
}
}
```

### מה יקרה בפעולה?

לפי כותרת הפעולה אנו מבינים כי הפעולה תבצע את ההוראות שבתוכה כשהכפתור `button1` יילחץ. לכן כל הוראה שנכתוב בתוכה תתבצע בעת הלחיצה על הכפתור. ההוראות יכולות לכלול (בין היתר) עדכון של מאפיינים (למרות שאת חלקם הגדרנו בטבלת המאפיינים): מאפיינים של הכפתור שנלחץ (הטקסט, צבע הרקע, תמונת רקע...), מאפיינים של כפתורים אחרים בטופס, מאפיינים של עצמים אחרים בטופס, מאפיינים של הטופס עצמו, וכו'.

דוגמה נוספת:

כאשר נלחץ פעמיים על ה-`PictureBox` שלנו (שה- `Name` שלו הוא `pictureBox1`), תיפתח לנו הפעולה הבאה:

```
private void pictureBox1_Click(object sender, EventArgs e)
{
}
}
```

גם כאן ההוראות שנכתוב בתוך הפעולה יתבצעו כאשר `pictureBox1` יילחץ. נדגיש שההוראות שיתבצעו הן ההוראות שכתבנו בין הסוגריים המסולסלים ( { } ) – בתוך

הבלוק.



**הגדרתו המדויקת של בלוק (Block) היא:** כל קטע קוד הנכתב בתוך סוגריים מסולסלים ( { } ) נקרא בלוק. הבלוק משמש לכתובת מספר רב של הוראות והוא גם פותח וסוגר מחלקות ופעולות.

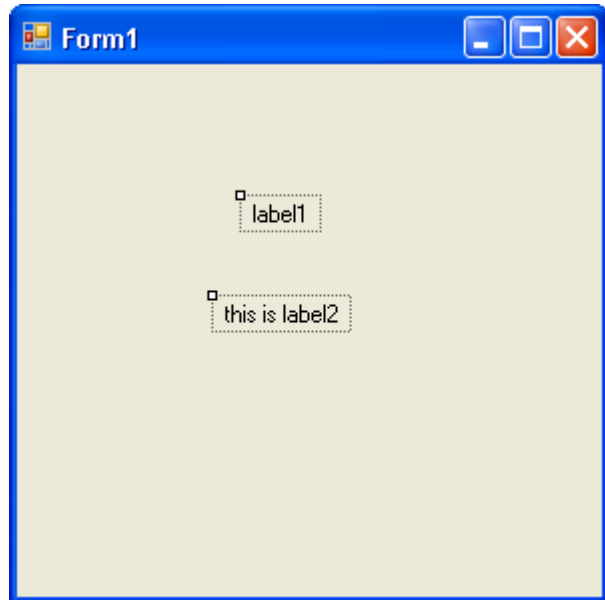
נזכור כי מעתה והלאה כאשר נלחץ פעמיים על עצם כלשהו, תיפתח לנו פעולה בקוד התכנות שלנו. הפעולה תתאים לאירוע (`Event`) שהכי סביר שיקרה בעצם (למשל: בכפתור- לחיצה עליו). בפעולה זו נוכל לכתוב קוד כרצוננו, והיא תבוצע כשהאירוע יקרה.

## תוויות - Labels ( לא חובה )

נוסיף מרכיב חדש למצבור המרכיבים שהכרנו עד עכשיו : Label- תווית. התוויות מציגות טקסט למשתמש, אך לא מאפשרות לו להוסיף טקסט בעצמו. בנוסף לכך, גבולות התווית אינם מוגדרים ואינם נראים בטופס הגמור (אלא רק בשלב העיצוב), והם נקבעים בהתאם לאורך וגודל הטקסט שבתווית. ה-labels נראים כך :



טופס גמור



טופס בשלבי עיצוב

כפי שניתן להסיק, ה-label יכול להיות ריק (ואז לא נראה אותו כלל בטופס הגמור עד שיהיה בו טקסט) ויכול להכיל טקסט. חשוב לשים לב שלא ניתן לסמן טקסט שבתווית.

את ה-label, כמו כל עצם/מרכיב/קומפוננט אחר נגרור מן ה-Toolbox אל הטופס שלנו.

### למה נשתמש בתוויות?

אנו נשתמש לכל אורך השנה בתוכניות שלנו בתוויות מפני שהן אלו שיצרו לנו תוכנית ברורה וקריאה למשתמש בה וינחו אותו לבצע את הפעולות הדרושות בתוכנית. לולא התוויות לא היינו יודעים כיצד עלינו לפעול בתוכנית והיו יכולות לקרות בעיות הבנה, כך כשיש לנו מעין "הערות", התכנות והשימוש יהיו יעילים ונוחים יותר.

## פתרון משימת ההיכרות

לאחר המידע שצברנו במשימה זאת נפתור את תוכנית ההיכרות שלנו על ידי שימוש בכפתורים, בתמונות ובמאפיינים תוך שילובם בקוד התוכנה שלנו.

### שלבם לביצוע התוכנית:

1. נגדיר את המאפיין Text של הטופס (שמהווה את הכותרת שלו) להיות "Hello, world!".
2. נגרוך כפתור לטופס.
3. נגדיר את התכונות הראשוניות לכפתור:
  - א. נסתכל בטבלת המאפיינים (Properties).
  - ב. ניכנס למאפיין Name, ונשנה את השם button1 לשם משמעותי כראות עיניכם.
  - ג. ניכנס למאפיין Text, ונכתוב: "קוראים לי...".
  - ד. ניכנס למאפיין BackColor, ונגדיר את צבע הרקע של הכפתור לאדום.
  - ה. ניכנס למאפיין ForeColor, ונגדיר את צבע הטקסט שבכפתור לכחול.
4. **כתיבת הקוד:**
  - א. לוחצים פעמיים על הכפתור, ואז יופיע לנו המקום שנועד לקוד.
  - ב. מוצאים את הפעולה שמתאימה ללחיצה על הכפתור הראשון, ובתוכה נכתוב את ההוראות הנדרשות למילוי המשימה. כאן יש לשנות את הטקסט של הכפתור בעת הלחיצה. נבצע זאת על ידי הפקודה:

```
this.שם_הכפתור.Text = "Ron";
```

כך, ברגע שנלחץ על הכפתור יופיע הטקסט "Ron" במקום "קוראים לי...".

עתה נסו לבצע את משימת ההיכרות בתוכנה ולראות כי הנכם מבינים את המשימה!





## משימות לעבודה עצמית (או...שעורי בית)



### משימת מנורות

פתח ויישם טופס המכיל שני כפתורים ושני pictureBox (שנו את השמות של העצמים!). המשימה היא:

1. נגדיר שהטקסט של כפתור אחד יהיה "הדלקה", ואילו הטקסט של הכפתור השני - "כיבוי".
2. נייבא שתי תמונות: של מנורה כבויה ל-pictureBox1 ושל מנורה דולקת ל-pictureBox2.
3. לפני הפעלת התוכנית, נגדיר ש-pictureBox1 יהיה בלתי נראה (Visible = false). כך נראה בתחילת הריצה של התוכנית את התמונה של המנורה הכבויה.
3. לאחר הפעלת התוכנית כאשר נלחץ על כפתור ההדלקה, נשנה את מאפייני ה- Visible כך שהמנורה הכבויה תיעלם ובמקומה תופיע המנורה הדולקת.
4. כשנלחץ על כפתור הכיבוי, נשנה את מאפייני ה- Visible כך שהמנורה הדולקת תיעלם ובמקומה תופיע המנורה הכבויה שוב.



### משימת מרוץ מכוניות

פתח ויישם טופס המכיל שלושה כפתורים ו-pictureBox (שנו את השמות של העצמים!). המשימה היא:

1. לפני הפעלת התוכנית נעצב את הכפתורים כך שייראו כמו רמזור המזניק מכונית במרוץ. כמו כן, נייבא אל ה-pictureBox תמונה שהיא אנימציה של מכונית שנוסעת. נגדיר שה-pictureBox יהיה בלתי נראה.
2. בעת הפעלת התוכנית, מה שנרצה שיקרה זה שהמשתמש ילחץ על שלושת כפתורי הרמזור (מלמעלה עד למטה). כל לחיצה תעלים את הכפתור שנלחץ. לחיצה על הכפתור האחרון תגרום גם להופעת תמונת האנימציה.
3. השתמשו היטב בבמאפיין Enabled כדי לוודא שהמשתמש יוכל כל פעם ללחוץ על כפתור אחד בלבד, ושזה יהיה הכפתור המתאים.



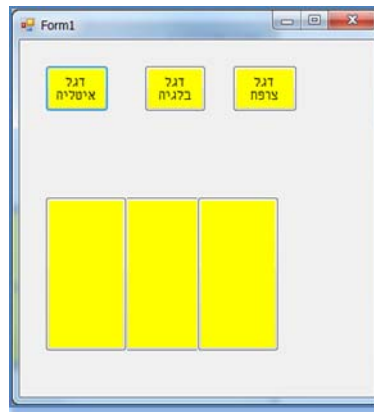
### משימת סמיילי

פתח ויישם טופס המכיל כפתורים היוצרים פרצוף (שני כפתורים לעיניים, כפתור לאף וכפתור לפה), ותווית (label) במקום כלשהו (שנו את השמות של העצמים!). המשימה היא:




1. הכפתורים, לפני הפעלת התוכנית, יהיו ללא טקסט וללא צבע (הצבע יהיה Transparent). המאפיין Text של התווית יכיל את הערך "smiley", אך היא תהייה בלתי-נראית בתחילה.
2. בעת הפעלת התוכנית כאשר נלחץ על אחת העיניים, ייצבעו כפתורי העיניים בכחול. בנוסף, משנלחצה אחת העיניים, התכנית תשנה את המאפיין Enabled של שני הכפתורים המתאימים כך שלא ניתן יהיה ללחוץ פעם נוספת על העיניים.
3. כשנלחץ על האף, הוא ייצבע בצהוב. כמו כן, טקסט הכפתור של האף יהיה "nose" בעקבות הלחיצה.
4. לחיצה על כפתור הפה תשנה את צבעו לאדום ואת צבע הרקע של הטופס ל"צבע גוף" (צבע הפרצוף). יתרה מזאת, התווית תהפוך לנראית.

## משימת דגלים

פתח ויישם טופס שיהיו בו שישה כפתורים- כולם ברקע צהוב (אל תשכחו לשנות את השמות של העצמים!). על כפתור אחד יהיה כתוב "דגל איטליה", על הכפתור השני יהיה כתוב "דגל בלגיה" ועל הכפתור השלישי יהיה כתוב "דגל צרפת". כפתורים מס' 4, 5 ו 6 יהיו ללא כיתוב ויוצבו צמודים אחד לשני. הם יהוו את הדגל. הטופס ייראה כך:



**המשימה:** לחיצה על אחד משלושת כפתורי הדגלים תציג את דגל המדינה על גבי כפתורים 4-6 לפי הפירוט הבא:

- דגל איטליה** - הוא פסי אורך של הצבעים (משמאל לימין) ירוק, לבן ואדום, 
- דגל בלגיה** - הוא פסי אורך של הצבעים (משמאל לימין) שחור, צהוב ואדום, 
- דגל צרפת** - הוא פסי אורך של הצבעים (משמאל לימין) כחול, לבן ואדום, 

לחיצה על כפתור של דגל תציג את הדגל בכפתורים 4-6, בהתאם למסכים הבאים:





## ד. משימת רמזור :

את שיעור זה (מס 2) נפתח במשימה הכוללת חזרה על החומר שנלמד בשיעור הקודם

במשימה זו אנו נבנה רמזור הבנוי מ-3 כפתורים (Buttons), אשר יפעל כמו רמזור בכביש.

על מנת לעשות זאת נזכר **כיצד הרמזור פועל בחיי היום-יום :**

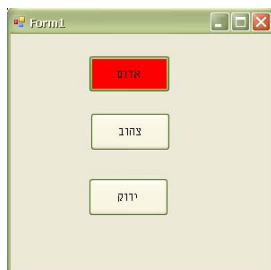
1. הרמזור נמצא במצב אדום- למכוניות אסור לנסוע.
2. הרמזור הופך לצהוב- המכוניות צריכות להיות בכוננות לנסיעה.
3. הרמזור הופך לירוק- המכוניות רשאיות לנסוע.
4. הרמזור הופך לצהוב+ירוק – המכוניות צריכות להיות בכוננות עצירה
5. הרמזור הופך לאדום - למכוניות אסור לנסוע.
- ו.....חוזר חלילה

**נרצה שהרמזור בתוכנית שלנו יפעל כך :**

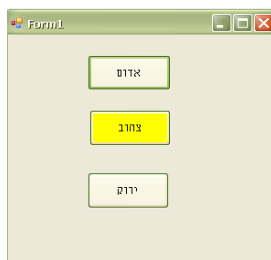
1. נכניס שלושה כפתורים : אדום (עצור), צהוב (התכונן) וירוק (סע).
2. אדום פועל מיד עם תחילת התוכנית, לכן רקע הכפתור הראשון יהיה מראש אדום.
3. כאשר נלחץ על הכפתור האדום, הכפתור הצהוב יפעל והכפתור האדום יכבה.
4. כאשר נלחץ על הכפתור הצהוב, הכפתור הירוק יפעל והצהוב יכבה.
5. כאשר נלחץ על הכפתור הירוק, יידלק גם הכפתור הצהוב
6. כאשר נלחץ על הכפתור הצהוב, ייכבו הכפתורים הצהוב והירוק ויידלק הכפתור האדום ו...חוזר חלילה

**הטופס ייראה כך לאחר הרצת התוכנית :**

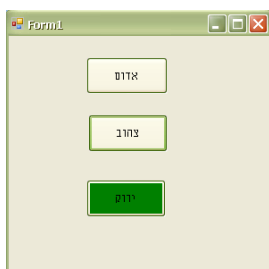
1. **מצב התחלתי :**



2. **אחרי לחיצה על כפתור אדום :**



3. **אחרי לחיצה על כפתור צהוב :**





כעת פתרו את שאלת הרמזור לפי מה שלמדתם במשימת ההיכרות!



**התלמידים (ואנחנו) נשים לב שקיימת פה !!!**

המשתמש יכול ללחוץ על כל כפתור כרצונו בלי קשר לסדר הארועים שקיים ברמזור.

נפתור את הבעיה בעזרת המאפיין Enable המאפשר (כן/לא) ללחוץ על הכפתור.



נמשיך ונפתור את שאלת הרמזור לפי מה שנלמד בתחילת השעור !



**שוב שנים לב שנוצרה פה בעיה חדשה !!!**

נתנו לכפתור הצהוב לעשות שתי משימות שונות בעת לחיצה. אבל... כיצד הוא יידע האם להדליק

את הכפתור הירוק או את הכפתור האדום?

נוכל לפתור את הבעיה בעזרת תנאים.





## ה. הוראת הפקודה תנאי פשוט

יש מקרים בהם נרצה שהוראות מסוימות יתבצעו רק אם תנאי כלשהו יתקיים.  
 על מנת להמחיש זאת ניתן דוגמה מחיי היום-יום:  
 אם נרוויח בעבודה 100 ש"ח לפחות, נלך ללונה פארק (לפיכך, ההליכה ללונה פארק מותנית).

ננסח בעברית את הוראת התנאי: בשפת ה-C# מבנה התנאי הוא:

<pre>         (ביטוי בוליאני) if         {             הוראות לביצוע;         }         else         {     </pre>	<p>אם (ביטוי בוליאני) אז                  הוראות לביצוע                  אחרת                  הוראות לביצוע</p>
---	--

נסביר עתה על מרכיבי ההוראה:

1. המילים אם-if, אחרת-else הן מילים שמורות כלומר אין להשתמש בהן חוץ מבכתיבת תנאי!
2. ביטוי בוליאני (מהטיפוס bool) הוא בעל אחד מבין שני הערכים: true (אמת) או false (שקר). אם ערכו הוא true, יבוצעו ההוראות בבלוק ה-if בלבד.
3. משמעות ה-else היא שכאשר ערך הביטוי הבוליאני הוא false, יבוצעו ההוראות שבתוך הבלוק של ה-else בלבד.

### טבלה של אופרטורים בוליאניים להשוואה

דוגמה לביטוי בוליאני	בשפת C#	בכתיב מתמטי	במילים
$a == b$ בודק האם הערכים של a ו-b שווים: אם כן, ייתן ערך אמת; אם לא, ייתן ערך שקר.	==	=	שווה
$a != b$ בודק האם הערכים של a ו-b שונים: אם כן, ייתן ערך אמת; אם לא, ייתן ערך שקר.	!=	≠	שונה
$3 < 6$ מאחר ש-3 קטן מ-6, הערך של הביטוי יהיה אמת.	<	<	קטן
$3 \leq 6$ מאחר ש-3 קטן או שווה ל-6, ערך הביטוי יהיה אמת.	<=	≤	קטן או שווה
$a + b > c$ בודק האם הערך שהוא הסכום של a ושל b גדול מהערך של המשתנה c: אם כן, ייתן ערך אמת; אם לא, ייתן ערך שקר.	>	>	גדול
$a + b \geq c * 2$ בודק האם הערך שהוא הסכום של ערכי המשתנים a ו-b גדול או שווה לערך שהוא המכפלה של הערך של c ב-2: אם כן, ייתן ערך אמת; אם לא, ייתן ערך שקר.	>=	≥	גדול או שווה

## כללים לשימוש בהוראה התנאי:

1. בסוף הביטוי הבוליאני לא מופיע הסימן ; וגם לא בתוך הסוגריים.
2. הביטוי הבוליאני חייב להופיע בתוך סוגריים.
3. כאשר סדרת ההוראות לביצוע כוללת יותר מהוראה אחת יש לתחום את ההוראות בסוגריים מסולסלים ( { } ) - בלוק הוראות. אם קיימת הוראה אחת בלבד לביצוע, ניתן להשמיט את הסוגריים של הבלוק. למרות זאת, גם כאשר לא נדרש בלוק, הוא מקל על קריאת התוכנית, ועל כן נעדיף להשתמש בו.
4. החלק של ה- else הוא אופציונאלי- ניתן להשמיט אותו. לעומת זאת, החלק של ה- if הוא חובה בתנאים פשוטים.
5. הביטוי הבוליאני שבסוגריים של התנאי יכול להופיע באחת מבין 3 הצורות הבאות:
  - א. ערך בוליאני מפורש: כלומר, true או false.נעיר שאין היגיון להשתמש בערך מפורש כביטוי הבוליאני בתנאים. הסיבה לכך היא שאם נכתוב true, אז ההוראות שבבלוק ה- if יבוצעו תמיד (ואז אפשר לכתוב אותן כרגיל, ללא התנאי), וההוראות שבבלוק ה- else לא יבוצעו בכלל (ואז אפשר פשוט לא לכתוב אותן).
- ב. משתנה בוליאני: למשתנה בוליאני יש את אחד מבין שני הערכים true ו- false. לכן ניתן להשתמש גם בו בשביל התנאי. נדגים זאת (נניח לשם הדוגמה שקיים לנו משתנה בשם b שהוגדר ואותחל לטיפוס bool):

```
if (b)
{
    הוראות;
}
else
{
    הוראות;
}
```

- במקרה זה, אם הערך של b הוא true, יבוצעו ההוראות שבבלוק ה- if; ואם הערך של b הוא false, יבוצעו ההוראות שבבלוק ה- else.
- ג. ביטוי עם אופרטור השוואה: ביטוי בוליאני באמצעות האופרטורים שבטבלה אשר נמצאת בעמוד הקודם. ההוראות שבבלוק ה- if יבוצעו אם ערך הביטוי הוא true. אם ערך הביטוי הוא false, יבוצעו ההוראות שבבלוק ה- else.
  - ד. ביטוי בוליאני מורכב: נלמד על כך בהמשך.

## כיצד נשתמש בתנאים ברמזור שלנו?

בפעולה של הכפתור הצהוב נבדוק, באמצעות תנאי, האם הכפתור הירוק דולק. כך נוכל להחליט האם יש להדליק את הכפתור הירוק או את הכפתור האדום.

```
if (this.button2.BackColor == Color.Green)
```

(

**קעת ניתן להמשיך ולבצע את משימת הרמזור - בעזרת תנאים!**





## 1. שימוש במשתנה int

במשימת הרמזור השתמשנו במאפיין חיצוני, צבע הכפתור, בכדי להבדיל בין המצבים השונים של הכפתור הצהוב.

לא תמיד נרצה להיות תלויים במשהו חיצוני ולכן נשתמש במשהו פנימי (משתנה) כדי להגדיר את המצב שלנו (אחרי אדום או אחרי ירוק).

נבקש מהתלמידים פתרון למשימה:

מן הסתם נקבל את התשובה (או דומה לה) נשמור 1 עבור מצב אדום ו-2 עבור מצב ירוק.

ו...זה הזמן להסביר מה זה משתנים פשוטים, מסוג int.

### משתנים

תוכניות מחשב מתבססות על משתנים ועל הצבת ערכים למשתנים. משתנה הוא בעצם תא בזיכרון של המחשב, ובתוך התא מאחסנים ערך. למשל ההוראה:  $a = 5$ ; בתוכנית מחשב משמעותה - קח את הערך 5 והכנס אותו למשתנה בשם a.



#### המשתנה מורכב מכמה מרכיבים:

1. **טיפוס המשתנה** - טיפוס המשתנה מגדיר את הערכים שמותר להציב במשתנה - מספר, תו, מילה או משפט.
2. **שם המשתנה** - שם המשתנה יכול להיות אות או מילה, להן ניתן להוסיף ספרות.
3. **ערך המשתנה** - לכל משתנה יש ערך. המשתנה יכול להכיל אך ורק ערך אחד, ואת הערך הזה אפשר לשנות במהלך התוכנית, בהמשך נראה כיצד.

#### חוקים להגדרת שם משתנה:

- המשתנה יכול להכיל אותיות אנגליות, מספרים וקווים תחתונים בלבד!
- התו הראשון במשתנה יכול להיות אך ורק אות או קו תחתון ולא יכול להסתיים בקו תחתון.
- C# רגישה לגודל אותיות, כלומר: המשתנה hello שונה מהמשתנה Hello או HELLO.
- לא ניתן להשתמש במילים שמורות של C# בתור משתנים. לדוגמה: לא תוכל ליצור משתנה בשם int כי זאת מילה שמורה.
- נהוג לבחור שם בעל משמעות על מנת להתמצא טוב יותר בתוכנית. לדוגמה: משתנה שערכו הוא מידע על זמן נקרא לו time.
- נהוג להתחיל שם משתנה באות קטנה. כאשר שם משתנה מורכב מכמה מילים נהוג להתחיל כל מילה חדשה באות גדולה על מנת שתהיה הפרדה בין המילים.
- ברוב המקרים בהם רוצים להגדיר טיפוס מסוג שלם, נהוג להשתמש ב-int (ולא ב-long).
- ברוב המקרים בהם רוצים להגדיר טיפוס מסוג ממשי, נהוג להשתמש ב-double (ולא ב-float).

## למה צריכים את זה?

אנו נעזרים במשתמשים על מנת לאחסן נתונים הדרושים לביצוע פעולות באלגוריתם באופן נוח ונגיש. המשתנים נשמרים בזיכרון המחשב ויש לנו האפשרות לגשת אליהם באמצעות שם המשתנה- המזהה. חיוניות המשתנה מתבטאת בכך שבמקרה ואין נתונים ממשיים בתכנית, עדיין ניתן לכתוב את הפעולות המתבצעות על המשתנה ורק במהלך הרצת התכנית להקנות לו ערך ממשי.

## מבנה של הגדרת משתנים

### מבנה בסיסי של הגדרת משתנה

כל משתנה יוגדר בצורה הבאה:

**שם\_המשתנה טיפוס\_המשתנה ;**  
יש לבחור שם משתנה חוקי על-פי הכללים שבעמוד הקודם. כמו כן, יש להציב נקודה-פסיק בסוף הגדרת המשתנה, שכן זו הוראה ככל ההוראה אחרת.

דוגמאות:

```
int a;  
bool b;  
char c;  
double d;  
string s;
```

### אתחול משתנה

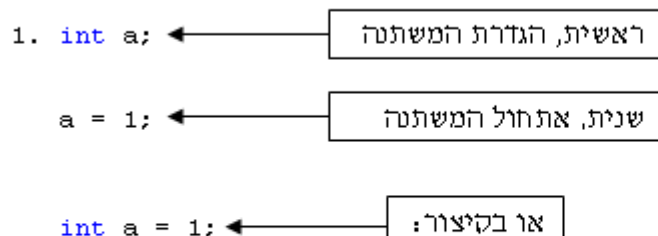
אתחול המשתנה פירושו **הקניית הערך הראשוני למשתנה**. כלומר, שמירת ערך מסוים בזיכרון המחשב (הערך הוא כרצוננו ובהתאם למטרת האלגוריתם). הגישה אל ערך זה תהיה דרך שם המשתנה שהגדרנו. אתחול משתנים נעשה ע"י הסימן שווה (=), שנקרא גם **סימן ההשמה** (משום שמשמימים ערך למשתנה).

יש לשים לב להקנות למשתנה ערך בהתאם לטיפוס המשתנה, אחרת תתרחש שגיאת תחביר. כמו כן, לא ניתן לאתחל משתנה שלא הוגדר לפני כן.

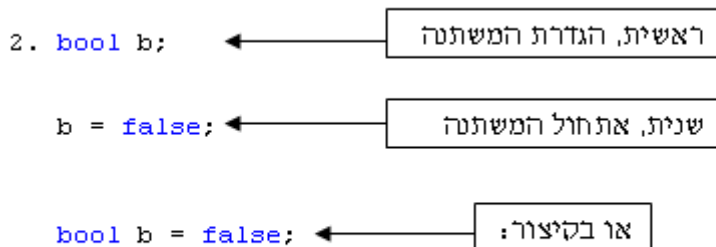
כל משתנה יאותחל בצורה הבאה:

**ערך = שם\_המשתנה ;**  
ניתן לחבר את הוראת הגדרת המשתנה להוראות האתחול שלו כך:  
**ערך = שם\_המשתנה טיפוס\_המשתנה ;**

דוגמה:



בדוגמה זו יצרנו משתנה מטיפוס שלם (int) ששמו a וערכו 1. למרות הבדל הכתיבה, לשתי הצורות משמעות זהה.



בדוגמה השנייה יצרנו משתנה מטיפוס בוליאני (`bool`) ששמו `b` וערכו `false`. גם כאן לשתי צורות הכתיבה משמעות שווה.

**הערה:** ניתן להגדיר ולאתחל כמה משתנים מאותו הטיפוס באותה השורה. המשתנים השונים יופרדו בפסיקים. נעשה זאת במבנה הבא:

... ; `ערך_לאתחול2 = שם_משתנה2, ערך_לאתחול1 = שם_משתנה1 טיפוס`

ניתן לא לתחל חלק מהמשתנים (או את כולם), אבל אז האתחול יגיע בשלב מאוחר יותר בתוכנית. דוגמאות:

`int num1 = 1, num2 = 2, num3 = 3, num4 = 4;`

`string str1 = "First", str2, str3 = "Third", str4="Fourth", str5;`

`char c1, c2, c3, c4, c5, c6, c7, c8;`

## טיפוס המשתנים `int`:



`int` הוא טיפוס המאפשר למשתנה לאחסן ערך של מספר שלם (ללא ספרות אחרי הנקודה) לדוגמה: 5, 1000, 234, וכו'. הטווח של המספרים שמשתנה מסוג `int` יכול להכיל הוא: מ-  $-2^{31} - 2,147,483,648$  ועד  $+2^{31} + 2,147,483,648$ .

**הגדרת משתנה `int` ב-`C#`:** ראשית יש להגדיר את המשתנה שבו נרצה להשתמש. כמו שלמדנו, הגדרת המשתנה תתבצע באחת משתי הדרכים:

- הגדרת המשתנה ללא אתחול. כך עדיין אין בו ערך ממש. אם נרצה שיהיה בו ערך, נצטרך להשים את ערך זה אל המשתנה. לדוגמה: `int a;`
- הגדרת המשתנה ואתחולו. לדוגמה: `int a = 57;`

לאחר שהגדרנו משתנה ואתחלנו אותו, נוכל להשים לו ערך חדש: ערך מפורש מטיפוס `int`, משתנה מטיפוס `int` או ביטוי שטיפוסו הוא `int`.

## שינוי ערך של משתנה / השמת ערך חדש למשתנה

לאחר שהגדרנו משתנה ואתחלנו אותו, אפשר לשנות את הערך שלו (לא לחינם משתמשים בשם **משתנה**). נדגיש שוב שמשתנה יכול להכיל ערך אחד בלבד בזמן נתון. נשנה ערך של משתנה על-פי המבנה הבא:

ערך\_חדש = שם\_המשתנה ;

**שימו לב:**



בהוראות השמה תמיד הערך של אגף ימין יועתק ויושם למשתנה שבאגף שמאל.

לשם הקניית הערך משתמשים גם פה, כמו באתחול משתנים, בסימן ההשמה (=). מיותר לציין שאי אפשר לשנות ערך של משתנה שלא הוגדר. נדגיש שהערך החדש חייב להתאים לטיפוס המשתנה. הערך החדש יופיע באחת מ-3 הצורות:

1. ערך קבוע ומפורש: ניתן כמה דוגמאות:

- בטיפוס int קיימים הערכים המפורשים 5, 7, 29, 131, וכו'. עבור משתנה a שהוגדר ואותחל לטיפוס int אפשר לכתוב, למשל, `a = 131;`
- בטיפוס char קיימים הערכים 'a', 'd', 't', וכו'. עבור משתנה c שהוגדר ואותחל לטיפוס char אפשר לכתוב, למשל, `c = 'a';`
- בטיפוס bool קיימים שני הערכים true ו- false. עבור משתנה b שהוגדר ואותחל לטיפוס bool אפשר לכתוב, למשל, `b = true;`
- בטיפוס string קיימים הערכים "hi", "C#", "Hello, world!", וכו'. עבור משתנה s שהוגדר ואותחל לטיפוס string אפשר לכתוב, למשל, `s = "C#";`

2. משתנה: נניח שקיימים שני משתנים שהוגדרו ואותחלו לטיפוס מסוים. שמות המשתנים הם num1 ו-num2. ניתן להשים משתנה אחד לשני. לדוגמה, ההוראה הבאה תכניס את הערך שב- num1 אל num2: `num2 = num1;`

3. ביטוי: קומביניציה של ערכים מפורשים ומשתנים מהטיפוס המתאים שיחברו זה אל זה באופרטור מתאים. נשתמש ב**סוגריים** כדי להבהיר למחשב את סדר הביצוע בחישוב הביטוי. לדוגמה, ב- int: נניח שיש לנו משתנים a, b ו-c שהוגדרו ואותחלו לטיפוס int. ניתן לבצע, למשל, את ההשמות הבאות:

```
a = b + 3;  
c = (a - 5) / 3;  
b = ((a % 6) * b) + (c / 8);
```

**שימו לב:**



בהשמת ביטוי למשתנה קודם מחושב ערך הביטוי, ואז ערך זה מושם למשתנה. נזכור: **החישוב קודם להשמה**.



### היכן נגדיר משתנים בקוד שלנו?

משתנים "מתים" כשיוצאים מהבלוק שבו הם הוגדרו. נזכור: **משתנה "מת" בסוף הבלוק שבו הוא "נולד"**. במסגרת הלימוד אנו נתכנת רק בסביבת מחלקת הטופס שלנו. על כן, אם נצטרך לגשת למשתנה מסוים מתוך שתי פעולות (לכל אחת מהן יש בלוק שונה), לא נגדיר את המשתנה בבלוק של אחת מבין הפעולות, מפני שאז לא נוכל לגשת אל המשתנה מבלוק הפעולה האחרת.

נפתור את הבעיה על-ידי הגדרת המשתנה בבלוק מחלקת הטופס (ולא בבלוק פנימי יותר). בלוק הטופס כולו מכיל את הבלוקים של כל הפעולות. לעומת זאת, משתנים שנצטרך לצורך פעולה מסוימת בלבד, נגדיר בבלוק של אותה הפעולה.

### כיצד אנו משתמשים במשתנים בשביל לשדרג את הרמזור?

נגדיר משתנה מסוג `int`.

בתחילה, כשהטופס עולה, בפעולה `Form_Load`, נשים בו ערך 1.

בכל פעם שנלחץ הכפתור האדום נשים במשתנה ערך 1.

בכל פעם שיילחץ הכפתור הירוק נשים במשתנה ערך 2.

באמצעות המשתנה הזה ובאמצעות התנאים שכבר למדנו, נוכל להגדיר קוד מתאים לפעולת הלחיצה על הכפתור הצהוב.

כעת ניתן להמשיך ולבצע את משימת הרמזור- בעזרת משתנה `int` שייבדק בפקודת



התנאי !

## ז. שימוש במשתנה `bool`

נעלה לדיון את השאלה האם השימוש במשתנה `int` הוא הפתרון האידאלי.

הרי אנו בודקים ערך 1 או 2 אבל...מבחינה עקרונית יכול להיות בו גם 3, 4 או כל מספר אחר בו אנו לא מטפלים.

נגיע עם התלמידים לצורך להשתמש במשתנה עם 2 מצבים בלבד – משתנה `bool` בו יש ערך אמת (`true`המסמן שהגענו מהכפתור האדום) או שקר (`false`המסמן שהגענו מהכפתור הירוק).

ו...זה הזמן להסביר משתנים מסוג `bool`.

## הגדרת משתנים ב-C#, חזרה והרחבה

- לכל משתנה כמו שהזכרנו קודם חייב להיות טיפוס, שם וערך.
- לפני שימוש במשתנה בתוכנית מחשב חייבים להגדיר את הטיפוס שלו. אנו נלמד תחילה 3 טיפוסים משתנים- int, double, bool אך יחד עם זאת נציג את כל סוגי המשתנים הנפוצים:

טיפוס המשתנה	הסבר	תחום ערכים	הצהרה ב-C#
<b>int</b>	ערכים שלמים	$-(2^{31}) - 2^{31}$	int x; למשל: 3, 10, 20040
<b>long</b>	ערכים שלמים	$-(2^{63}) - 2^{63}$	long y; למשל: 214923485
<b>float</b>	ערכים ממשיים	דיוק של עד 7 ספרות אחרי הנקודה	float a; למשל: 1.3334232
<b>double</b>	ערכים ממשיים	דיוק של עד 15-16 ספרות אחרי הנקודה	double b; למשל: 3.31111113
<b>char</b>	תווים (ספרה, אות, סימן)	אין תחום	char c; למשל: '!', '3', 'a'
<b>string</b>	מחרוזות (רצף) תווים	אין תחום	string s; למשל: "halo"
<b>bool</b>	אמת או שקר	true / false	bool b;

### טיפוס המשתנה bool :



bool הוא טיפוס של משתנה שיש לו שני ערכים אפשריים: true (אמת) או false (שקר). הגדרת משתנה מטיפוס bool תיעשה כך:

```
bool x = false;
```

או:

```
bool x = true;
```

ניתן כמובן להפריד את ההגדרה והאתחול לשתי שורות נפרדות. גם עבור משתנה מסוג bool שהוגדר ואותחל בשלב מוקדם יותר בקוד, נוכל להשים ערך חדש: ערך בוליאני מפורש, משתנה בוליאני או ביטוי ובוליאני (על ביטויים בוליאניים נלמד בהמשך).

לדוגמה, נבחר במשתנה בוליאני. בכל פעם שיילחץ הכפתור האדום, ערך המשתנה יהיה "אמת"; ובכל פעם שיילחץ הכפתור הירוק, ערך המשתנה יהיה "שקר".





## משימה עצמית / שעורי בית

### משימת מרינה

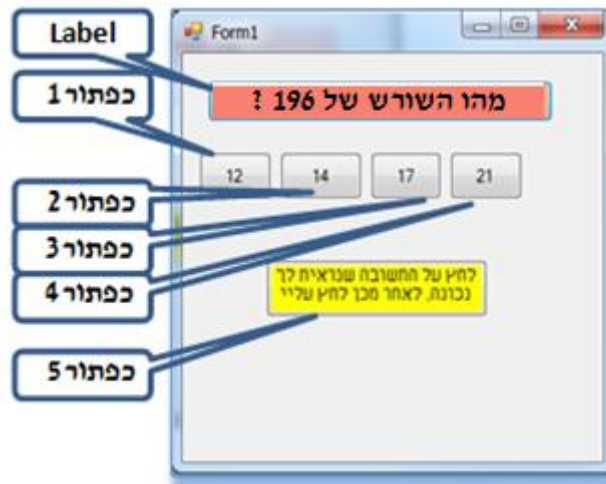
במרינה הדמיונית בירושלים נקבע הכלל: על כל 5 ספינות שנכנסות, נכנס אופנוע ים אחד. אופנוע ים לא יכול להיכנס כל עוד לא נכנסו 5 ספינות; ואם נכנסו 5 ספינות, לא תוכל להיכנס ספינה נוספת עד שייכנס אופנוע ים. פתח ויישם טופס המכיל שלושה כפתורים. הטופס שלנו נפתח על-ידי עובדי הנמל בעת פתיחת המרינה. המשימה היא:

1. נגדיר שהטקסטים של הכפתורים יהיו: "ספינה", "אופנוע ים", "סוף יום"; כמו כן הכפתור של האופנוע ים יוגדר להיות בלתי לחיץ (לא נכנסו עוד 5 ספינות).
2. לאחר שכפתור הספינה ילחץ 5 פעמים, כפתור זה ייהפך לבלתי לחיץ (לא ניתן להכניס ספינות נוספות עד שיגיע אופנוע ים). באופן טבעי, הכפתור של האופנוע ים ייהפך ללחיץ.
3. לחיצה על כפתור "אופנוע הים תגרום תהפוך אותו להיות בלתי לחיץ, ואת כפתור הספינה - ללחיץ.
4. שלבים ג'-ד' חוזרים חלילה עד שבשלב בלשהן נלחץ כפתור סוף היום.
5. לחיצה על כפתור סוף היום מביאה לכך שכל הכפתורים הופכים לבלתי לחיצים.

### משימת מחשבון:

המשימה: נכתוב שאלה כלשהי עם ארבע אפשרויות, כשרק אחת מהן נכונה. אם המשתמש לוחץ על התשובה הנכונה, נכתוב לו "תשובה נכונה"; ואם הוא טועה, נכתוב "תשובה לא נכונה, נסה שוב".

שלבים לכתיבת התוכנית:



1. נבנה טופס עם 5 כפתורים ותווית אחת. נגדיר את מאפייניהם לפי התרשים הבא:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    int x;

    private void button2_Click(object sender, EventArgs e)
    {
        x = 1;
    }

    private void button3_Click(object sender, EventArgs e)
    {
        x = 2;
    }

    private void button4_Click(object sender, EventArgs e)
    {
        x = 3;
    }

    private void button5_Click(object sender, EventArgs e)
    {
        x = 4;
    }

    private void button6_Click(object sender, EventArgs e)
    {
        if (x == 2)
        {
            button6.Text = "תשובה נכונה";
        }
        else
        {
            button6.Text = "תשובה לא נכונה, נסה שוב";
        }
    }
}
```

2. label- מציינת שאלה כלשהי למשתמש.

3. הכפתורים שהמשתמש ילחץ עליהם הם כפתורים 2-6, לכן יש ללחוץ על כל אחד מהם, כדי שתיפתח שורת קוד.

4. נגדיר משתנה x מסוג int.

5. לכל אחת מארבע התשובות האפשריות שנמצאות בכפתורים 2-5 ניתן ערך שונה ל- x (התשובה הנכונה נמצאת בכפתור מספר 3 וערך ה- x שלו הוא 2 - ראו בתוכנית שמוצגת משמאל).

6. בכפתור 5 נכניס תנאי. אם x שווה ל- 2 (התשובה הנכונה), אז הטקסט של כפתור זה ישתנה ל-"תשובה נכונה", אחרת הטקסט ישתנה ל-"תשובה לא נכונה, נסה שוב".

## שאלה

מה יקרה בתוכנית זו אם המשתמש ילחץ על כפתור הבדיקה לפני שלחץ על אחד מהכפתורים שבהם יש את התשובות השונות? ענו בעצמכם! איך פותרים את זה ??