Microsoft Small Basic
סמול בייסיק מאת מיקרוסופט

מבוא לתכנות

פרק 1

הקדמה

סמול בייסיק ותכנות מחשבים

תכנות מחשבים הוא התהליך שבו יוצרים תוכנות מחשבים ע"י שימוש בשפות תוכנה. כשם שאנו מדברים ומבינים עברית, ערבית, אנגלית או שפה אחרת, כך מחשבים הן מכונות היכולות לבצע תוכניות הכתובות בשפות מסוימות. שפות אלו נקראות שפות תכנות. בתחילה היו שפות תכנות בודדות והן היו קלות ללימוד ולתפיסה. אבל, ככל שמחשבים ותוכנות נהיו יותר ויותר מתוחכמים, גם שפות התוכנה התפתחו במהירות ונעשו מסובכות יותר.
כתוצאה מכך רוב שפות התכנות המודרניות והרעיונות שמאחוריהן, הפכו למאתגרות למדי עבור מתחילים. עובדה זו התחילה להרחיק אנשים מלימוד או מהתנסות בתכנות מחשבים.

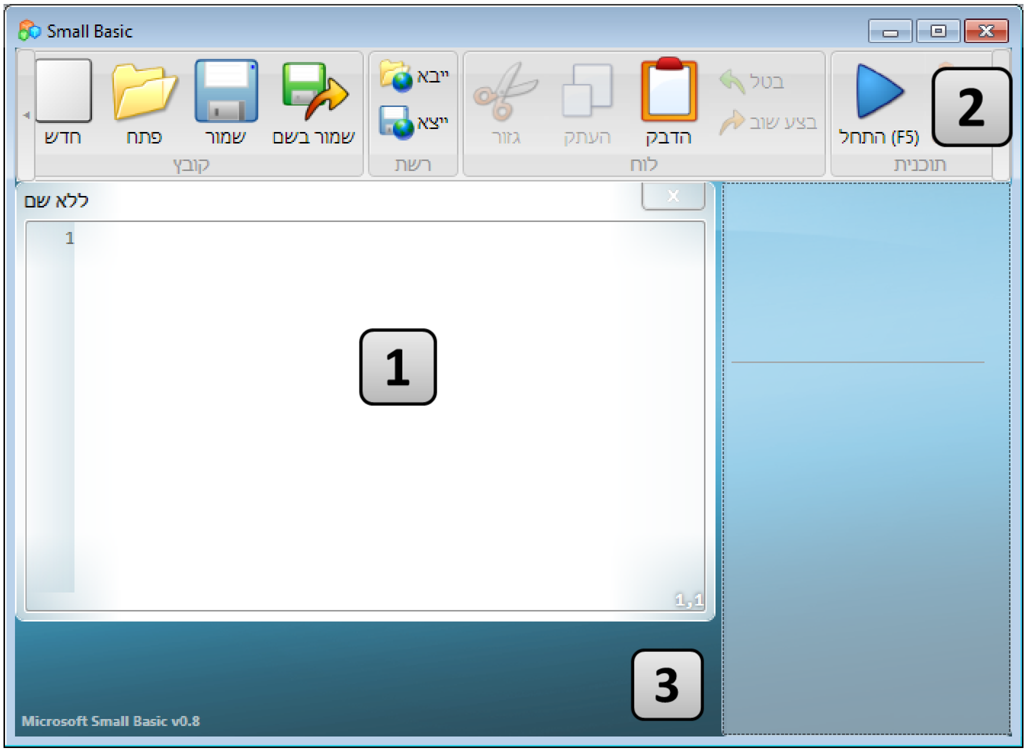
Small Basic[1] היא שפת תכנות המתוכננת כך שהתכנות יהפוך עבור מתחילים לקל, נגיש ובעיקר - מהנה. הכוונה של Small Basic היא לשמש כשער לעולם המופלא של תכנות מחשבים.

סביבת העבודה של Small Basic

נתחיל במבוא קצר לסביבת העבודה של Small Basic. בפעם הראשונה כשמפעילים את Small Basic רואים חלון שנראה כמו זה המופיע באיור הבא:

---

[1] הערה בקשר לגרסה העברית של מדריך זה: השם של Small Basic ניתן לתרגום מילולי לעברית כ: שפה בסיסית קטנה או שמא בייסיק בקטנה. חברת מיקרוסופט, שפיתחה שפה זו, רואה בה גם מוצר תוכנה ובמקרה שכזה נוהגת להשאיר את השם המקורי באנגלית וכך ננהג בהמשך המדריך. המתרגם נעזר במילון של חברה זו עבור תרגום מונחים מקצועיים, בשמות מקובלים בתעשיית התוכנה המקומית ובמילונים נוספים. בעברית ישנו הבדל ניכר בין פנייה לקורא בלשון זכר או נקבה, משום כך השתדלתי להשתמש בלשון רבים שבה ההבדלים ניכרים פחות, זאת לנוחות כלל המשתמשות והמשתמשים. למרות שהמדריך הוא בעברית, שפת התכנות עצמה משתמשת באותיות ובמילים פשוטות של השפה האנגלית, כך שידע ראשוני באנגלית יוכל לעזור. המדריך בעברית מוקדש **לאחיה**, שהספיק לעשות רק כמה צעדים ראשונים בתכנות מחשב עם לוגו (ראו פרק 8) והתקשה בהיעדר סביבה ומדריך בשפתו.



איור 1 – הסביבה של Small Basic

זוהי סביבת העבודה של Small Basic, בה נכתוב ונריץ את התוכניות. לסביבה זו יש כמה מאפיינים ייחודיים המסומנים באיור במספרים.

עורך הקוד המסומן ע"י [1], הוא המקום בו נכתוב את התוכניות ב- Small Basic. גם כשפותחים תוכנית לדוגמה או תוכנית שנשמרה קודם, היא תופיע בעורך הקוד. כך אפשר לשנות את התוכנית ולשמור אותה לשימוש עתידי.

אפשר גם לפתוח ולעבוד בו-זמנית על יותר מתוכנית אחת. כל תוכנית שעובדים עליה מוצגת בחלון נפרד של עורך הקוד. החלון שבו נמצאת התוכנית שעליה עובדים כרגע, נקרא *החלון הפעיל*.

סרגל הכלים המסומן ע"י [2], מאפשר להפעיל פקודות על החלון הפעיל או על סביבת העבודה כולה. נכיר את הפקודות השונות תוך כדי העבודה.

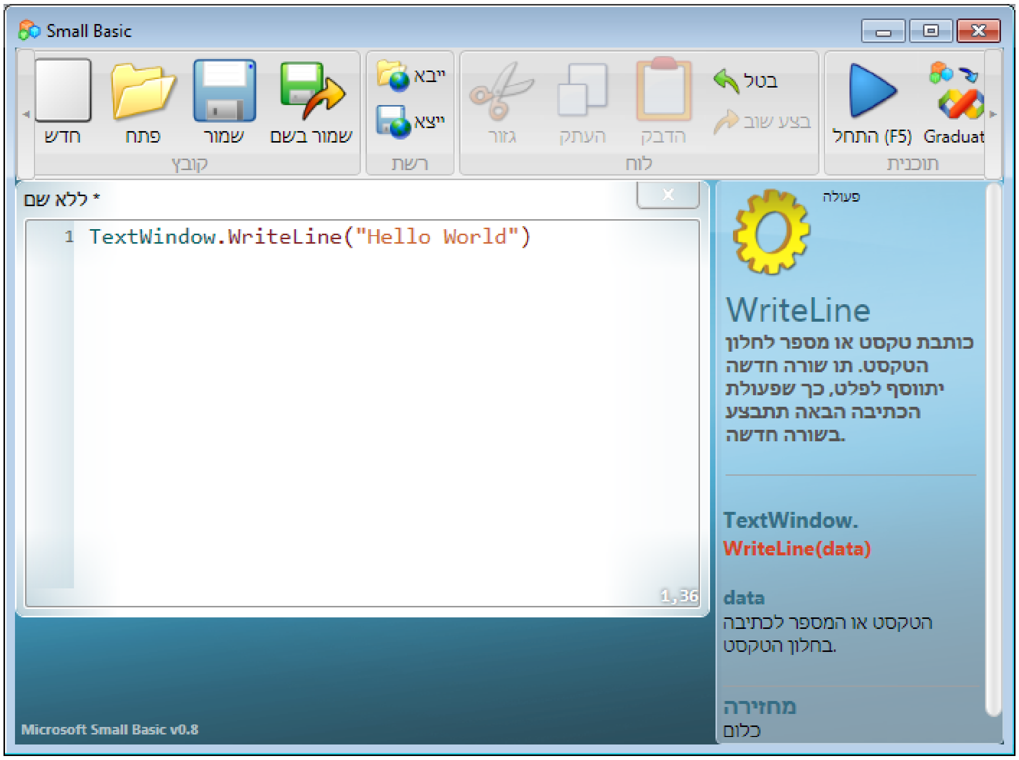
ה**משטח** המסומן ע"י [3] הוא המקום עליו נמצאים החלונות של עורך הקוד.

התוכנית הראשונה שלנו

עכשיו כשאנחנו מכירים את סביבת העבודה של Small Basic, נתקדם ונתחיל לתכנת בעזרתה. כפי שכבר צוין עורך הקוד הוא האזור בו אנו כותבים את התוכניות, אז בואו נכתוב את השורה הבאה בעורך הקוד (כן-כן, עכשיו הזמן לעבור ל- Small Basic ולהתחיל לעבוד! כדאי מאוד גם לנסות להקליד את התוכנית לבד)

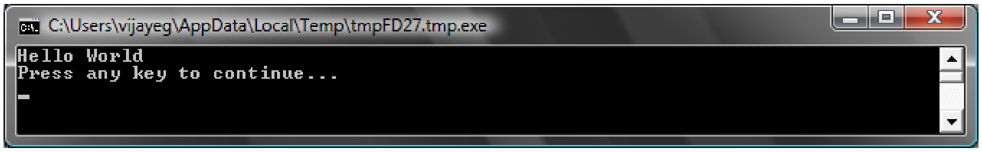
```
TextWindow.WriteLine("Hello World")
```

זוהי התוכנית הקטנה הראשונה שלנו ב- Small Basic. אם הקלדתם אותה נכון, זה צריך להיראות כמו באיור הבא:



איור 2 – תוכנית ראשונה

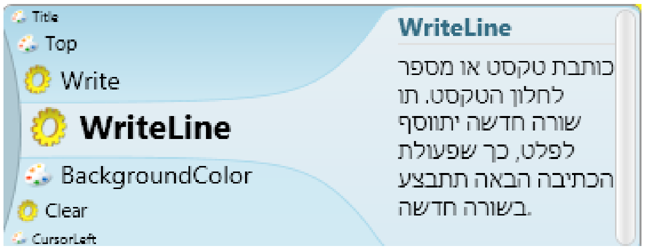
עכשיו כשהקלדנו את התוכנית החדשה שלנו, בואו ננסה להריץ אותה ולבחון מהי התוצאה. אפשר להריץ את התוכנית ע"י לחיצה על הכפתור "*התחל*" בסרגל הכלים או ע"י שימוש במקש קיצור F5 במקלדת. אם הכול תקין, התוכנית שלנו אמורה להתבצע והתוצאה אמורה להיות דומה לחלון המוצג למטה.



איור 3 – הפלט של התוכנית הראשונה

ברכות! כתבתם והרצתם תוכנית ראשונה ב- Small Basic. אמנם קטנה מאוד ופשוטה, אך בהחלט צעד גדול לקראת היותכם מתכנתי מחשבים אמיתיים! נשאר לנו להבין מה קרה פה – מה בדיוק רצינו שיקרה בתוכנית ואיך הדברים התרחשו. בפרק הבא ננתח את התוכנית שכתבנו, כך שנוכל להבין מה התרחש (בשורה השנייה במסך שנפתח כשהרצנו את התוכנית, כתוב שיש להקיש על מקש כלשהו כדי לסגור חלון זה ולחזור ל- Small Basic).

תוך כדי הקלדת התוכנית, בוודאי שמתם לב שהופיע תפריט מוקפץ עם רשימת פריטים (איור 4). זה נקרא "intellisense" והוא עוזר לנו לכתוב תוכניות מהר יותר. אפשר לעבור על רשימה זו בעזרת מקשי החיצים למעלה\למטה וכאשר מגיעים לפריט שמעוניינים בו, מקישים על Enter כדי להכניס את הפריט לתוכנית. בנוסף, מקבלים הסבר על הפריטים השונים.



איור 4 - Intellisense

שמירת התוכנית שלנו

אם תרצו לסגור את Small Basic ולהמשיך לעבוד אחר כך על אותה התוכנית שהקלדתם, אפשר לשמור אותה. כדאי לשמור את התוכנית מידי פעם גם תוך כדי עבודה, כך שלא יאבד מידע אם המערכת תיפול, או אם תהיה בעיה באספקת החשמל וכדומה. אפשר לשמור את התוכנית הנוכחית על ידי לחיצה על הכפתור "שמור" בסרגל הכלים או בעזרת קיצור הדרך Ctrl+S (לחיצה על S תוך כדי שהמקש Ctrl לחוץ).

פרק 2

בואו נבין את התוכנית הראשונה שלנו

מהי בעצם תוכנית מחשב?

תוכנית היא רשימת הוראות לביצוע עבור המחשב. הוראות אלו, שלפעמים קוראים להן פשוט קוד, מתארות למחשב מה לבצע והמחשב מבצע אותן בדיוק רב. מחשבים יכולים לבצע הוראות אם הן נמסרות בשפות המובנות להם. שפות אלו נקראות שפות תכנות. ישנן שפות תכנות רבות ואחת מהן היא **Small Basic**.

ישנן שפות רבות לתכנות מחשבים. Java, C++, Python, VB, Scratch ועוד. כולן שפות תוכנה מודרניות המשמשות לפיתוח תוכנות מפשוטות ועד מורכבות.

דמיינו דו שיח ביניכם לבין חבר. הדיבור יכלול מילים היוצרות ביחד משפטים ובדרך זו עובר מידע ביניכם. בדרך דומה גם בתוכניות מחשב יש אוסף של מילים היוצרות ביחד משפטים וכך אנחנו מעבירים מידע למחשב. תוכניות הן קבוצות משפטים (לעיתים רק אחדות ולפעמים אלפים ויותר) שיש להן משמעות גם למתכנת שכתב אותן וגם למחשב שאמור לבצע אותן.

תוכניות של Small Basic

בתוכנית רגילה של Small Basic יש אוסף משפטים. כל שורה מייצגת בדרך כלל משפט העובר כהוראה למחשב. כאשר אנחנו מבקשים להריץ תוכנית ב- Small Basic, המחשב קורא את התוכנית והיא מתחילה להתבצע כבר מהשורה הראשונה. לאחר ביצוע השורה הראשונה, המחשב קורא את השורה השנייה ומבצע אותה וכך הלאה שורה אחרי שורה עד לסוף התוכנית.

בחזרה לתוכנית הראשונה

הנה שוב התוכנית הראשונה שכתבנו:

.

```
TextWindow.WriteLine("Hello World")
```

זוהי תוכנית פשוטה שיש בה *משפט* אחד בלבד. משפט זה אומר למחשב לכתוב לתוך חלון הטקסט שורה עם המילים **Hello World** (שלום עולם)[2].

הכוונה היא לומר למחשב לבצע:

```
כתוב Hello World
```

בוודאי שמתם לב שאת המשפט שכתבנו בתוכנית אפשר לחלק לקטעים קצרים יותר, בדומה למשפט המתחלק למילים. במשפט הראשון ישנם 3 חלקים נפרדים:

א) TextWindow – חלון הטקסט
ב) WriteLine – כתוב שורה
ג) "Hello World" – "שלום לעולם", הטקסט לכתיבה

הנקודה, הסוגריים והגרשיים הם פיסוק במשפט ונועדו לעזור למחשב לפרש או להבין את המשפט בצורה מדויקת.

סימני פיסוק כמו גרשיים, רווחים וסוגריים, הם בדרך כלל מאד משמעותיים בתוכניות מחשב. המיקום שלהם קובע את המשמעות המדויקת של המשפט.

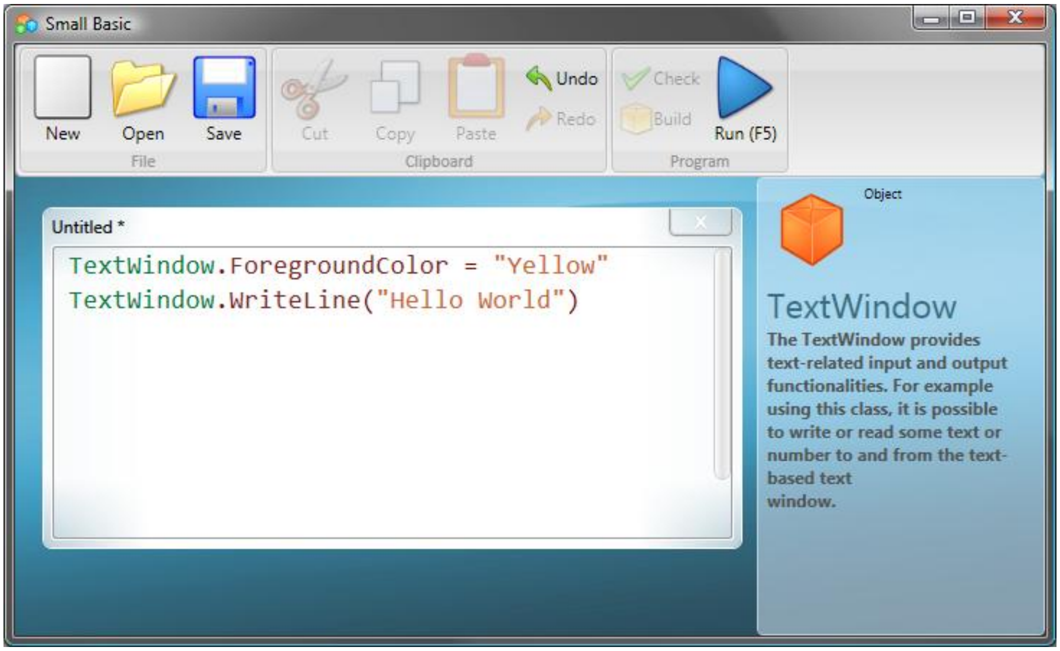
בואו ניזכר במסך השחור שהופיע כשהרצנו את התוכנית הראשונה. חלון זה נקרא חלון הטקסט (TextWindow) ולפעמים גם מסוף (Console). זהו המקום שאליו מגיעות התוצאות של הרצת התוכנית. בתוכנית שלנו **TextWindow** נקרא אובייקט. לרשותנו עומדים אובייקטים שונים שאפשר להשתמש בהם. לכל אובייקט ישנן פעולות שהוא מסוגל לבצע עבורנו ונלמד בהמשך על האובייקטים השונים והפעולות שלהם. בתוכנית הראשונה השתמשנו כבר בפעולה *WriteLine* של TextWindow. לאחר שם הפעולה הופיעו בתוכנית המילים "Hello World" בגרשיים ובסוגריים. הגרשיים מסמנים את כל הטקסט לכתיבה והסוגריים מסמנים שכל מה שבתוכם עובר לפעולה WriteLine לביצוע הכתיבה לחלון, כך שהמשתמש יראה את התוצאה. הטקסט עם הסוגריים נקרא *הקלט* לפעולה. ישנן פעולות המקבלות קלט אחד או יותר וישנן שלא מקבלות קלט בכלל.

התוכנית השנייה שלנו

עכשיו כשהבנו את התוכנית הראשונה, בואו נמשיך ונשכלל אותה ע"י הוספת צבעים.

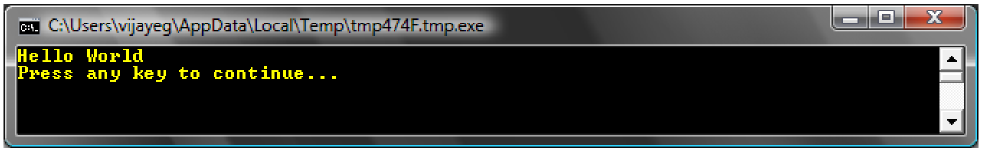
```
TextWindow.ForegroundColor = "Yellow"
TextWindow.WriteLine("Hello World")
```

[2] אפשר גם לכתוב בעברית "שלום עולם", אם כי ייתכן שיהיה צורך להתקין במערכת שלכם גופן עברית של MS-DOS כדי לראות את התוצאה, בהמשך כשנעבור לחלון גראפי בעיה זו תיפתר.



איור 5 – הוספת צבעים

כאשר נריץ תוכנית זו, נראה שהיא כותבת אותן המילים שהיו קודם בחלון הטקסט, רק שהפעם המילים מופיעות בצבע צהוב במקום בצבע האפור שהיה קודם.



איור 6 – שלום לעולם בצהוב

שימו לב למשפט החדש שהוספנו לתוכנית המקורית. הוא כולל מילה חדשה *ForegroundColor* (צבע קידמה) שאליו משווים את הערך "Yellow" ("צהוב"). המשמעות של המשפט אינה שהם שווים אלא שמבצעים בו *השמה* (הכנסה) של הערך צהוב לצבע הקידמה של חלון הטקסט. לעומת הפעולה WriteLine, המילה *ForegroundColor* לא קיבלה קלטים ואף לא השתמשה בסוגריים. במקום זאת מופיע אחריה סימן *שווה* (=) שלאחריו מילה. נגדיר ש- ForegroundColor הוא *מאפיין* של TextWindow. הנה רשימה של ערכים חוקיים עבור המאפיין ForegroundColor. נסו להחליף את "Yellow" באחד מהם, ואחר כך להריץ את התוכנית ולצפות בתוצאה – לא לשכוח את הגרשיים, הן פיסוק הכרחי.

```
Black
Blue
Cyan
Gray
Green
Magenta
Red
White
```

```
Yellow
DarkBlue
DarkCyan
DarkGray
DarkGreen
DarkMagenta
DarkRed
DarkYellow
```

פרק 3

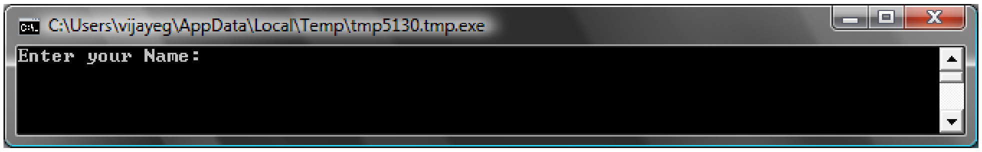
שימוש במשתנים

שימוש במשתנים בתוכנית שלנו

האם לא יהיה נחמד אם במקום כתיבת המשפט הכללי "שלום לעולם" ("Hello World"), התוכנית שלנו תגיד "שלום" עם שם המשתמש בתוכנית? כדי לבצע זאת אנחנו צריכים קודם כל לשאול את המשתמש מה שמו ואז לשמור את השם היכן שהוא. עכשיו נוכל לכתוב למסך "שלום" ולאחריו לכתוב את שם המשתמש. בואו נראה כיצד לעשות זאת:

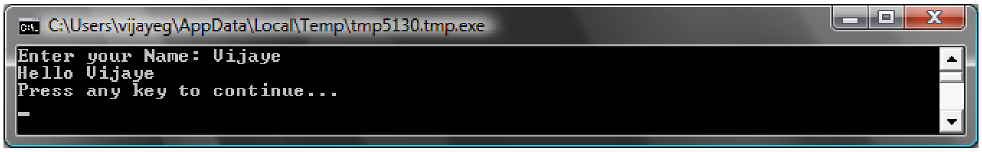
```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

כאשר נקליד ונריץ תוכנית זו, נקבל את הפלט הזה:



איור 7 – בקשת שם המשתמש

ואם נקליד את השם ונקיש על Enter, נקבל פלט כזה:



איור 8 – שלום מנומס

עכשיו אם נריץ שוב את התוכנית, התוכנית תשאל שוב מה שמנו. נוכל להדפיס שם אחר והפעם המחשב יגיד שלום למשתמש האחר.

ניתוח התוכנית

השורה שאולי תפסה את תשומת לבכם בתוכנית שעכשיו הרצנו, היא:

```
name = TextWindow.Read()
```

Read נראית כמו *WriteLine* רק שאין לה שום קלט. זוהי פעולה האומרת למחשב לחכות לקלט מהמשתמש על ידי שהמשתמש יקליד טקסט ויקיש על *Enter*. לאחר שהמשתמש ביצע זאת, הטקסט שהוקלד חוזר לתוכנית. לא משנה מה הקליד המשתמש, הטקסט הזה מאוחסן עכשיו ב*משתנה* שנקרא *name*. משתנה הוא דבר או מקום שאפשר לאחסן בו ערכים שונים באופן זמני, ולהשתמש בהם מאוחר יותר באמצעות השם שניתן להם. בשורת הקוד שנמצאת מעל, השתמשנו בשם *name*, כדי לשמור את שמו של המשתמש.

גם השורה הבאה מעניינת:

```
TextWindow.WriteLine("Hello " + name)
```

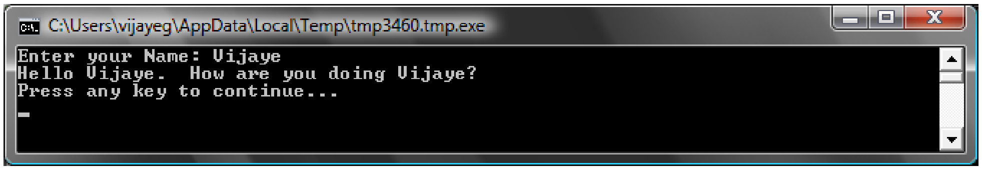
זהו המקום שבו אנו משתמשים בערך שאוחסן במשתנה name. אנחנו לוקחים את הערך שב- name, מחברים אותו ל- "Hello" בעזרת הסימן + ואז מעבירים זאת לחלון הטקסט לכתיבה.

> *Write כמו WriteLine היא פעולה של חלון הטקסט. Write מאפשר לכתוב לחלון כך שהטקסט הבא שייכתב יהיה עדיין באותה שורה בהמשך לטקסט הנוכחי.*

לאחר שמשתנה קיבל ערך, אפשר לחזור ולהשתמש בו כמה פעמים שנרצה. למשל אפשר לכתוב גם כך:

```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.Write("Hello " + name + ".  ")
TextWindow.WriteLine("How are you doing " + name + "?")
```

ואז יתקבל הפלט הבא:



איור 9 – שימוש חוזר במשתנה

כיצד נותנים שם למשתנה?

למשתנה יש שם המקושר אליו ומזהה אותו. ישנם כמה כללים פשוטים וגם הנחיות לבחירת שם למשתנה:

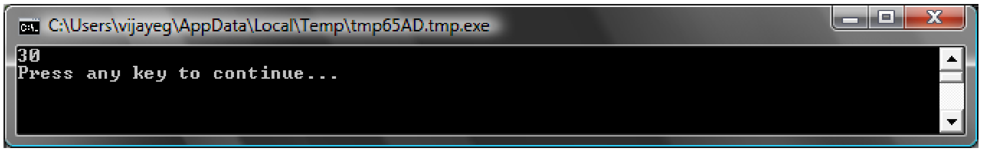
1. השם צריך להתחיל באות כאשר לאחריה יכול להיות רצף כלשהו של אותיות, ספרות וקווים תחתיים.
2. אסור שהשם יהיה מילה שמורה הקיימת בשפה כמו **if**, **for**, **then** או מילים אחרות שנלמד בהמשך.
3. כדאי לתת למשתנה שם משמעותי – מכיוון שהשם יכול להיות ארוך ככל שנרצה, אפשר לתת שם המציין באופן ברור את תפקיד המשתנה.

משחק במספרים

ראינו איך אפשר להשתמש במשתנה כדי לשמור את שם המשתמש. בתוכניות הבאות נראה איך נוכל לשמור מספרים במשתנים וגם לטפל בהם בצורות שונות. נתחיל בדוגמה מאד פשוטה:

```
number1 = 10
number2 = 20
number3 = number1 + number2
TextWindow.WriteLine(number3)
```

כאשר נריץ את התוכנית נקבל את הפלט הבא:



איור 10 – חיבור שני מספרים

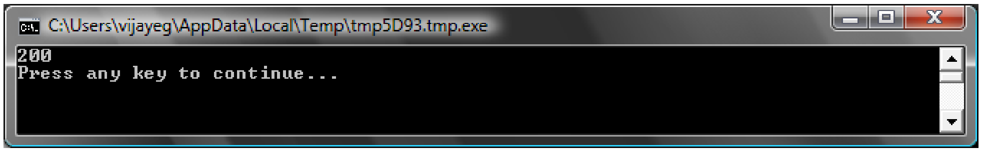
בשורה הראשונה אנו מבצעים הכנסה למשתנה **number1** של הערך המספרי 10. באופן דומה בשורה השנייה, אנו מבצעים השמה של 20 למשתנה **number2**. בשורה השלישית מחברים את ערכי המשתנים **number1** ו- **number2** ואת התוצאה אנו שמים במשתנה **number3**. כך שבמקרקה זה **number3** מכיל את הערך 30 וזה מה שנכתב לחלון הטקסט.

עכשיו, נשנה קצת את התוכנית ונצפה בתוצאות:

```
number1 = 10
number2 = 20
number3 = number1 * number2
```

```
TextWindow.WriteLine(number3)
```

התוכנית שלמעלה מכפילה את ערכו של **number1** ב- **number2** ושמה את התוצאה ב- **number3**. אפשר לראות את התוצאה הפעם למטה:



איור 11 – הכפלת שני מספרים

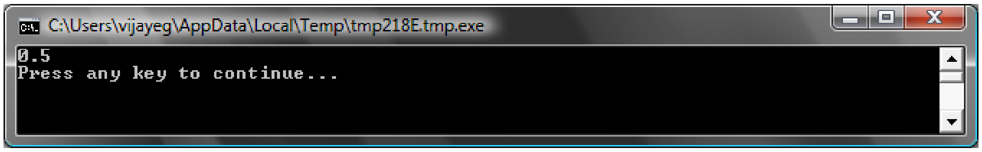
בצורה דומה אפשר להחסיר או לחלק מספרים, הנה דוגמה לחיסור:

```
number3 = number1 - number2
```

הסמל לחילוק הוא / והפעולה תיראה כך:

```
number3 = number1 / number2
```

תוצאת החלוקה תהיה:



איור 12 – חלוקת שני מספרים

ממיר מידות חום פשוט

בתוכנית הבאה נשתמש בנוסחה $°C = \frac{5(°F-32)}{9}$ כדי להמיר מעלות חום בפרנהייט למעלות בשיטת צלזיוס. ראשית, נקרא את הטמפרטורה בפרנהייט מהמשתמש ונשמור אותה במשתנה. ישנה פעולה המתאימה בדיוק למשימה הזו והיא **TextWindow.ReadNumber**.

```
TextWindow.Write("Enter temperature in Fahrenheit: ")
fahr = TextWindow.ReadNumber()
```

עכשיו כשיש לנו את הנתון בתוך משתנה, ניתן להמיר אותו בדרך הזו:

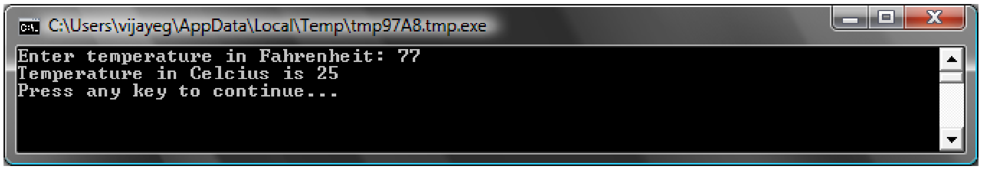
```
celsius = 5 * (fahr - 32) / 9
```

הסוגריים מסמנות שיש לחשב את החלק **fahr – 32** לפני שמכפילים ב- 5. כעת מה שנשאר זה לכתוב את התוצאה לחלון כדי שהמשתמש יצפה בה. אם נחבר את כל החלקים זה יראה כך:

שימו לב שהמספרים בדוגמה אינם מוקפים בסוגריים. יש צורך בסוגריים רק לטקסט.

```
TextWindow.Write("Enter temperature in Fahrenheit: ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("Temperature in Celsius is " + celsius)
```

והתוצאה תהיה:



איור 13 – המרת טמפרטורות

פרק 4

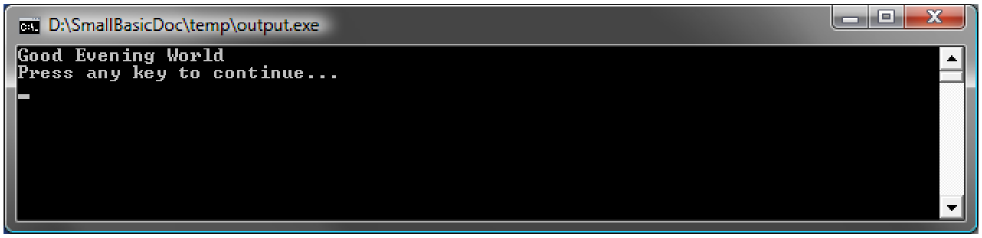
תנאים והסתעפויות

נחזור לתוכנית הראשונה שלנו, האם זה לא יהיה נחמד אם במקום להגיד תמיד *שלום עולם*, נוכל להגיד *בוקר טוב עולם* או *ערב טוב עולם* לפי השעה ביום? בתוכנית הבאה נדאג שעד השעה 12 בצהריים הפלט יהיה *בוקר טוב עולם* ולאחר מכן *ערב טוב עולם*.

```
If (Clock.Hour < 12) Then
      TextWindow.WriteLine("Good Morning World")
EndIf
If (Clock.Hour >= 12) Then
      TextWindow.WriteLine("Good Evening World")
EndIf
```

תלוי מתי נריץ את התוכנית בבוקר או בערב, נראה את אחד משני הפלטים הבאים:

איור 14 – בוקר טוב עולם



איור 15 – ערב טוב עולם

בואו ננתח את שלש השורות הראשונות של התוכנית. בוודאי ניחשתם שהתוכנית בודקת האם Clock.Hour קטן מ- 12 ובמקרה זה כותבת "בוקר טוב עולם". המילים **If**, **Then** ו- **Else** הן מילים מיוחדות שמשפיעות על ביצוע התוכנית. לאחר המילה **If** (אם) תמיד מופיע תנאי. במקרה זה התנאי הוא **(Clock.Hour< 12)**. שימו לב שהסוגריים הכרחיים כדי שהכוונה תהיה ברורה. לאחר התנאי מופיעה המילה **Then** (אז) שלאחריה תבוא הפעולה שיש לבצע כאשר התנאי מתקיים. לאחר הפעולה מופיעה המילה **EndIf** (סיום האם), המסמנת את סוף הביצוע המותנה.

ב- Small Basic אפשר להשתמש באובייקט השעון (Clock) כדי לדעת את התאריך והשעה העכשוויים. יש לו גם אוסף מאפיינים המאפשרים לקבל בנפרד את היום, החודש, השנה, השעה, הדקה או השנייה הנוכחיים.

בין ה- **then** וה- **EndIf**, אפשר לשים יותר מפעולה אחת והמחשב יבצע את כולן כאשר התנאי מתקיים. למשל אפשר היה לכתוב גם:

```
If (Clock.Hour < 12) Then
      TextWindow.Write("Good Morning. ")
      TextWindow.WriteLine("How was breakfast?")
EndIf
```

Else (אחרת)

אולי שמתם לב שבעצם התנאי השני בתוכנית שפתחה פרק זה הוא מיותר. הערך של השעה המתקבל מ- **Clock.Hour** יכול להיות או קטן מ- 12 או לא. כך שהיה אפשר לוותר על הבדיקה השנייה. במקרה כזה אפשר לקצר את שני משפטי ה- **if..then..endif** למשפט אחד על ידי שימוש במילה נוספת **else (אחרת)**.

אם נכתוב מחדש את התוכנית עם שימוש ב- **else**, היא תיראה כך:

```
If (Clock.Hour < 12) Then
      TextWindow.WriteLine("Good Morning World")
Else
      TextWindow.WriteLine("Good Evening World")
EndIf
```

תוכנית זו תתבצע בדיוק כמו הקודמת, דבר המלמד אותנו שיעור חשוב בתכנות מחשבים:

> *יש דרכים רבות להגיע אל אותה מטרה. לעיתים דרך אחת נראית יותר הגיונית מהשנייה והבחירה היא בידי המתכנת. תוך כדי כתיבת תוכניות רוכשים ניסיון ומתחילים להבחין בשיטות שונות, ביתרונות ובחסרונות שלהן.*

כניסה\הזחה

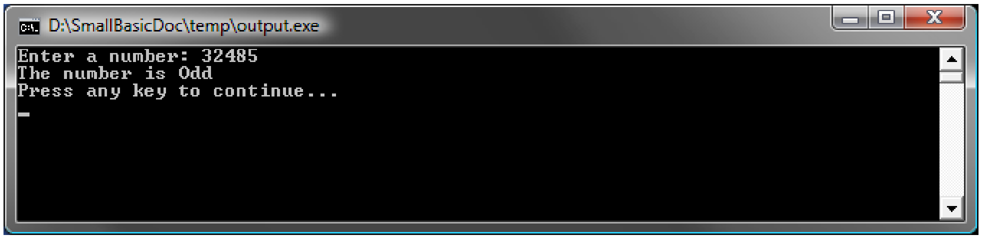
בדוגמאות שראינו אפשר לשים לב שהמשפטים שבין ה- *If*, *Else* ו- *EndIf* מוכנסים (מוזחים) מעט ימינה, כך שהמשפט מתחיל עם רווחים. ב- *Small Basic* כניסה זו אינה הכרחית. התוכנית תוכל להתבצע היטב גם ללא ההזחה, אך אנו עושים זאת כדי שהתוכנית והמבנה שלה יהיו יותר ברורים למתכנתים. לכן, הזחה של משפטים נחשבת הרגל טוב בתכנות.

זוגי או אי-זוגי

עכשיו כשיש לנו יכולת להשתמש במשפטי **If..Then..Else..EndIf**, בואו נכתוב תוכנית שמקבלת מספר ואומרת אם הוא זוגי (even) או אי-זוגי (odd).

```
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
      TextWindow.WriteLine("The number is Even")
Else
      TextWindow.WriteLine("The number is Odd")
EndIf
```

וכשנריץ את התוכנית, נוכל לקבל את הפלט הבא:



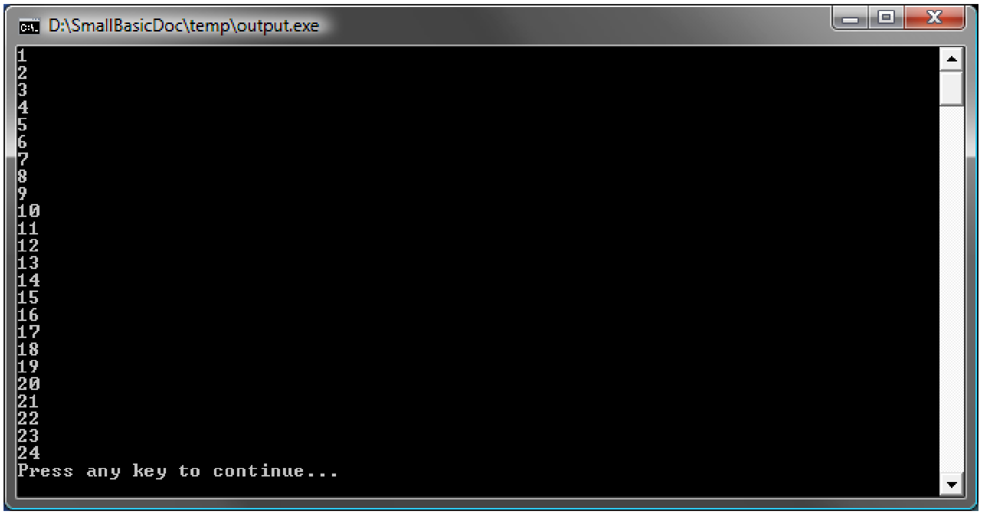
איור 16 – זוגי או לא?

בתוכנית זו השתמשנו בפעולה שימושית נוספת **Math.Remainder**. אולי כבר ניחשתם שפעולה זו מחלקת את המספר הראשון במספר השני ונותנת בחזרה את שארית החלוקה.

הסתעפויות

זוכרים שבפרק השני למדנו שתוכנית מחשב מתבצעת פקודה אחר פקודה מלמעלה למטה? מסתבר שיש גם פקודה מיוחדת שגורמת למחשב לקפוץ לפקודה אחרת שלא לפי הסדר הכתוב. בואו נבדוק את התוכנית הבאה:

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
  Goto start
EndIf
```



איור 17 – שימוש ב- Goto

בתוכנית שלמעלה, קבענו את ערכו של המשתנה i ל- 1. אח"כ רשמנו משפט המסתיים בנקודתיים (:)

```
start:
```

משפט זה נקרא *תווית* (*label*). תוויות הן כמו סימניות בתוכנית. אפשר לתת לסימנייה שמות לפי בחירתכם (במקרה זה start) ואין מגבלה על מספר הסימניות בתוכנית בתנאי שלכל אחת יש שם ייחודי (כמו כן בדרך כלל לא מומלץ להשתמש בתוויות מפני שמבנה התוכנית יכול להפוך ללא ברור – ובהמשך נלמד שיטות יותר מובנות לשינוי סדר המשפטים המתבצעים בתוכנית).

עוד שורה עם משפט מעניין היא זו:

```
i = i + 1
```

כאן, בצד ימין של המשפט, התוכנית מחברת 1 לערך שבמשתנה i ואז מעדכנת את ערכו של i בחזרה. אם למשל ערכו של i הוא 1 אז לאחר ביצוע המשפט ערכו 2.

לבסוף,

```
If (i < 25) Then
  Goto start
EndIf
```

זהו חלק התוכנית שבו אם ערכו של i נמוך מ-25, יש לחזור ולבצע את הפקודות החל מהתווית **start**.

ביצוע אינסופי

על ידי שימוש בהוראה **Goto** אפשר עקרונית לומר למחשב לבצע פעולות במספר פעמים כלשהו. למשל אפשר לשנות את התוכנית הקודמת כך שהיא תוכל לרוץ לתמיד. אפשר בכל זאת לעצור אותה אם לוחצים על כפתור הסגירה (X) שבפינה העליונה של החלון.

```
begin:
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
 TextWindow.WriteLine("The number is Even")
Else
 TextWindow.WriteLine("The number is Odd")
EndIf
Goto begin
```

איור 18 – זוגי או אי-זוגי רץ לנצח

פרק 5

לולאות

לולאות For

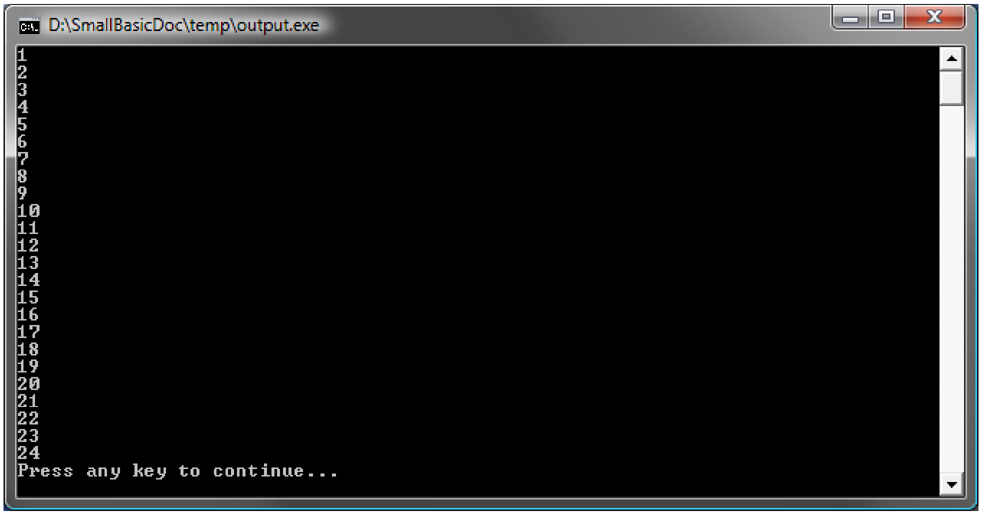
בואו נבחן שוב את התוכנית מהפרק הקודם:

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
  Goto start
EndIf
```

התוכנית כותבת מספרים לפי הסדר מ- 1 עד 24. התהליך של קידום הערך המאוחסן במשתנה, נפוץ מאד בתכנות, ולכן לשפות תכנות יש בדרך כלל שיטות נוחות יותר לביצוע משימה זו. את התוכנית שלמעלה אפשר לכתוב בצורה הבאה:

```
For i = 1 To 24
  TextWindow.WriteLine(i)
EndFor
```

והפלט יהיה:



איור 19 – שימוש בלולאת For

שימו לב שקיצרנו את התוכנית מ- 8 שורות ל- 4 שורות והיא עדיין מבצעת אותו הדבר. זוכרים שאמרנו שיש בדרך כלל מספר דרכים להגיע אל אותה התוצאה? זוהי דוגמה אחת לעניין.

במונחי תכנות, המבנה **For..EndFor** נקרא *לולאה*. הדבר מאפשר לנו לקחת משתנה, לתת לו ערכים של התחלה ושל סיום ולהשאיר לתוכנית את האחריות לקדם בשבילנו את המשתנה בין ערכים אלו. התוכנית מקדמת בכל פעם את ערך המשתנה ואם לא עברנו את ערך הסיום, מתבצעות ההוראות שבין **For** לבין **EndFor**.

מה יקרה אם נרצה לקדם את המשתנה ב- 2 במקום ב- 1 כדי להדפיס את כל המספרים האי-זוגיים למשל? גם במקרה זה אפשר להשתמש בלולאות.

```
For i = 1 To 24 Step 2
  TextWindow.WriteLine(i)
EndFor
```

איור 20 – רק האי-זוגיים

החלק של ה- **Step 2** בהוראה ה- **For** אומר לתוכנית לקדם את הערך של **i** בקפיצות של 2 במקום ב-1. אפשר לציין כל ערך לקידום על ידי שימוש ב- **Step**. אפשר אפילו לציין ערך שלילי ובכך לספור לאחור, כמו בדוגמה הבאה:

```
For i = 10 To 1 Step -1
  TextWindow.WriteLine(i)
EndFor
```

איור 21 – ספירה לאחור

לולאות While

יש סוג נוסף של לולאה הנקרא While (כל עוד), סוג זה הוא שימושי בעיקר כאשר לא ידוע מראש כמה פעמים יש לבצע את הלולאה. לעומת לולאת For שרצה מספר פעמים מוגדר מראש, לולאת ה- While רצה כל עוד הערך של תנאי מסוים הוא אמת. בדוגמה הבאה אנו מחלקים מספר כל עוד התוצאה גדולה מ-1.

```
number = 100
While (number > 1)
  TextWindow.WriteLine(number)
  number = number / 2
EndWhile
```

איור 22 – לולאת חלוקה

בתוכנית שלמעלה, המשתנה number מאותחל בערך 100 והלולאה רצה כל עוד הערך שלו גדול מ-1. בתוך הלולאה אנו כותבים את הערך של המשתנה ואז מחלקים אותו בשתיים. כתוצאה מכך הפלט הוא סדרה של מספרים שהולכים ומתחלקים בשתיים.

קשה יהיה לכתוב תוכנית כזו באמצעות לולאת For, מכיוון שמספר הפעמים שצריך להריץ את הלולאה תלוי בערך של המשתנה. במקרה כזה יהיה נוח יותר להשתמש בלולאת While משום שהיא בודקת בכל פעם אם התנאי לביצוע הלולאה מתקיים.

מעניין לציין שאפשר להפוך כל לולאת While למשפט If..Then עם Goto.

למעשה, באופן פנימי, התוכנית שאנחנו כותבים מתורגמת למשפטים כאלו, מכיוון שבדרך כלל חומרת המחשב יכולה לבצע ישירות רק אותם

למשל, התוכנית שלמעלה יכולה להיכתב כדלהלן והתוצאה תהיה זהה:

```
number = 100
startLabel:
TextWindow.WriteLine(number)
number = number / 2

If (number > 1) Then
  Goto startLabel
EndIf
```

פרק 6

מתחילים עם גרפיקה

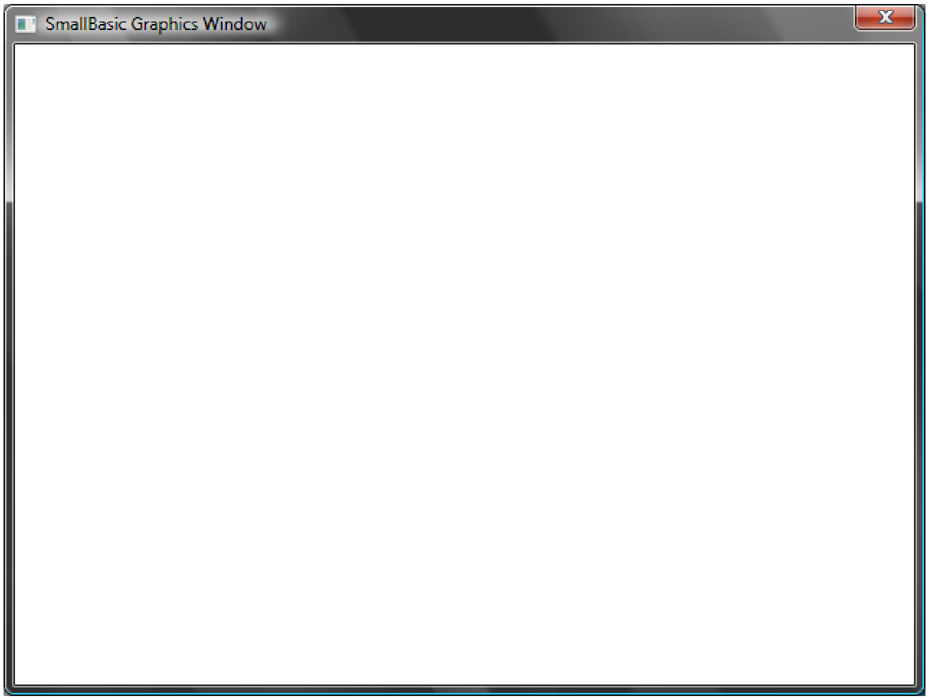
עד עכשיו בדוגמאות שראינו השתמשנו בחלון הטקסט כדי להכיר את הבסיס של שפת Small Basic. אבל, סביבה זו מכילה גם אפשרויות גראפיות שאותן נתחיל להכיר בפרק זה.

הצגת החלון הגראפי

כמו שהשתמשנו בחלון הטקסט כדי להציג טקסט ומספרים, כך נוכל להשתמש בחלון גראפי (**GraphicsWindow**) כדי לצייר דברים. נתחיל קודם כל בהצגת החלון הגראפי.

```
GraphicsWindow.Show()
```

כאשר נריץ את התוכנית, נשים לב שבמקום החלון השחור מקבלים חלון לבן בדומה למה שמופיע למטה. עדיין אין הרבה מה לעשות עם חלון זה, אך זהו הבסיס שאיתו נעבוד בהמשך הפרק. אפשר לסגור את החלון על ידי לחיצה על כפתור ה- X בפינה העליונה.



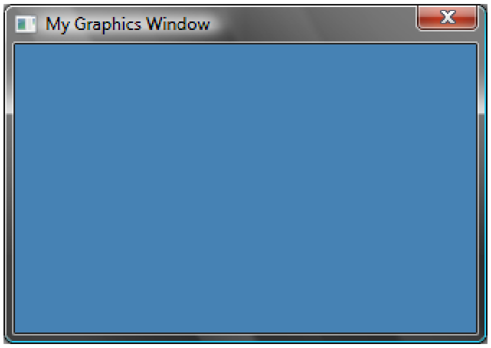
איור 23 – חלון גראפי ריק

הגדרת החלון הגראפי

החלון הגראפי מאפשר להגדיר מאפיינים שונים המשפיעים על דרך התצוגה שלו. אפשר לקבוע את הטקסט של הכותרת, את צבע הרקע ואת גודל החלון. בואו נכניס מספר שינויים כדי להכיר את החלון יותר טוב.

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "My Graphics Window"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

אפשר לשנות מאפיינים שונים של החלון, למשל את צבע הרקע אפשר לשנות לאחד מהצבעים המפורטים בנספח ב' בסופו של מדריך זה. נסו לשחק ולשנות מאפיינים אלו ולצפות בתוצאות.

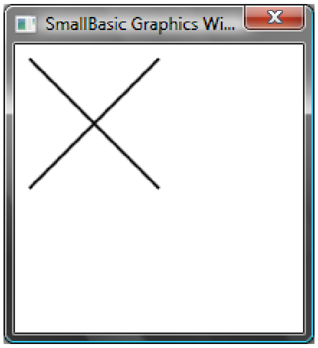


איור 24 – חלון גראפי מותאם

לצייר קווים

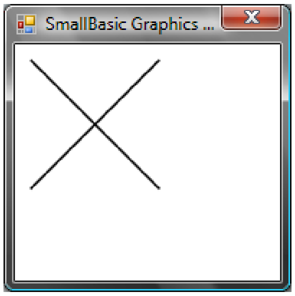
ברגע שהחלון הגראפי מוצג, נוכל לצייר עליו צורות, טקסט ואפילו תמונות. בואו נתחיל בציור מספר צורות פשוטות. הנה קטע מתוכנית המציירת שני קווים על החלון הגראפי.

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.DrawLine(10, 100, 100, 10)
```



איור 25 – שתי וערב

שתי השורות הראשונות קובעות את גודל החלון ושתי השורות הבאות מציירות את הקווים המצטלבים. שני המספרים הראשונים לאחר DrawLine מגדירים את קואורדינאטות ה- x וה- y של תחילת הקו, ושני האחרים את קואורדינאטות הסיום. שילמו לב שבגרפיקת מחשבים הקואורדינאטות (0,0) מציינות בדרך כלל את הפינה השמאלית העליונה של המסך (מבחינה מתמטית אומרים שכל החלון מופיע ברביע השני של מערכת הצירים).

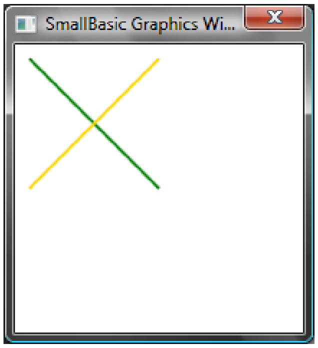


איור 26 – מערכת הצירים

אם נמשיך עם תוכנית הקווים, אפשר גם לשנות מאפיינים של הקו עצמו כמו צבע ועובי. דבר ראשון נשנה את צבע הקווים כמו בתוכנית הבאה:

במקום להשתמש בשמות הצבעים אפשר גם להשתמש בשיטת הצבעים ברשת (#RRGGBB). לדוגמא #FF0000 מסמן אדום, #FFFF00 מסמן צהוב וכדומה. פרטים נוספים בנספח ב' בסוף.

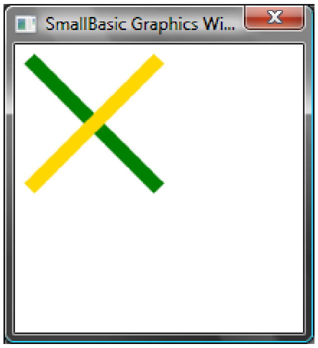
```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```



איור 27 – שינוי צבע קו

עכשיו נשנה גם את הגודל. בתוכנית הבאה אנחנו משנים את הרוחב ל- 10, במקום ברירת המחדל 1.

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenWidth = 10
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

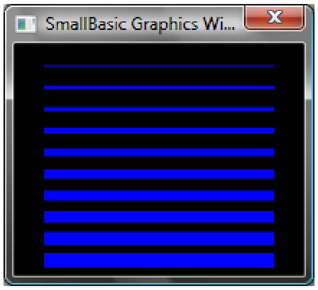


איור 28 – קווים עבים וצבעוניים

המאפיינים *PenWidth* ו- *PenColor* משנים את העט שבו משתמשים כדי לצייר קווים. שינוי זה משפיע לא רק על הקווים אלא גם על הצורות המצוירות לאחר שינוי מאפיינים אלו.

על ידי שימוש במשפטי לולאה מהפרקים הקודמים נוכל לכתוב תוכנית המציירת קווים ההולכים ונעשים עבים.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 160
GraphicsWindow.PenColor = "Blue"

For i = 1 To 10
  GraphicsWindow.PenWidth = i
  GraphicsWindow.DrawLine(20, i * 15, 180, i * 15)
endfor
```



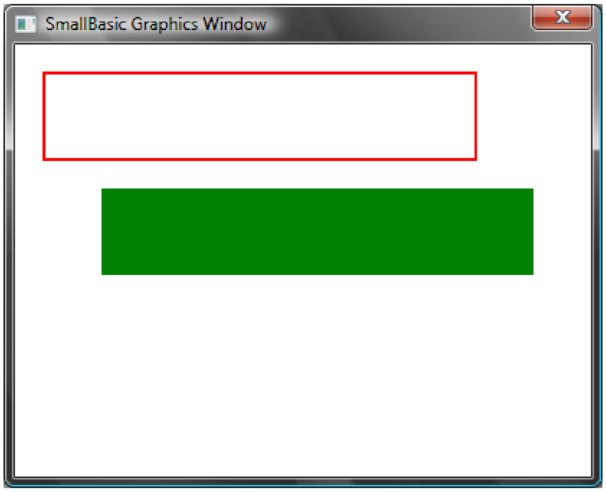
איור 29 – קווים בעובי שונה

החלק המעניין בתוכנית הוא הלולאה שבה אנחנו מגדילים את רוחב העט בכל סבב ומציירים קו חדש מתחת לקודם.

ציור ומילוי צורות

כאשר אנו רוצים לצייר צורות ישנן שתי פעולות: ציור (*Draw*) ומילוי (*Fill*). פעולות ציור מציירות את קווי המתאר של צורה באמצעות עט, לעומת פעולת המילוי המשתמשת במברשת. לדוגמה: בתוכנית הבאה, ישנם שני מלבנים, אחד מצויר באמצעות עט אדום ושני ממולא על ידי מברשת ירוקה.

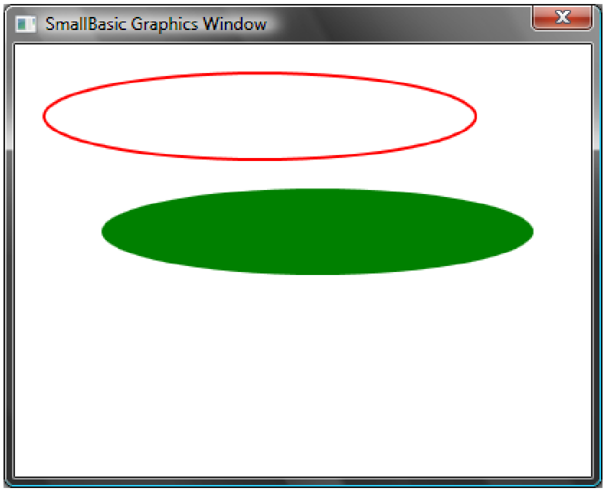
```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawRectangle(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillRectangle(60, 100, 300, 60)
```



איור 30 – ציור ומילוי

כדי לצייר מלבן או כדי למלא אותו יש צורך בארבעה מספרים. השניים הראשונים מציינים את הקואורדינאטות של הפינה השמאלית העליונה של המלבן. המספר השלישי מציין את הרוחב של המלבן והרביעי את הגובה שלו. למעשה אותם נתונים משמשים גם לציור אליפסות (סגלגלים) כפי שנראה בתוכנית הבאה:

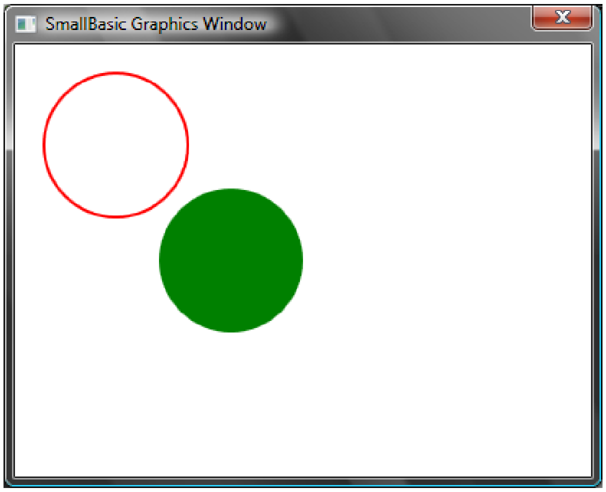
```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(60, 100, 300, 60)
```



איור 31 – ציור ומילוי אליפסות

אליפסות הן מקרה כללי של עיגולים. אם נרצה לצייר עיגולים, פשוט נציין רוחב וגובה שווים.

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 100, 100)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(100, 100, 100, 100)
```



איור 32 - עיגולים

פרק 7

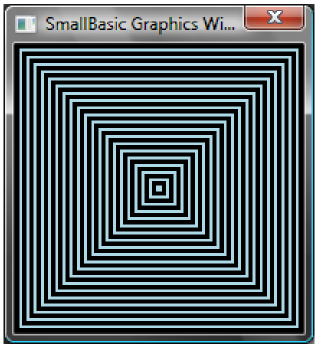
כיף עם צורות

בפרק זה נהנה ממה שלמדנו עד כה. נראה כמה דוגמאות הממחישות איך באמצעות מה שלמדנו עד עכשיו אפשר ליצור תוכניות מגניבות למדי.

מלבנים

בתוכנית הבאה, אנחנו מציירים בתוך לולאה סדרת מלבנים בגודל הולך ועולה.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
  GraphicsWindow.DrawRectangle(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

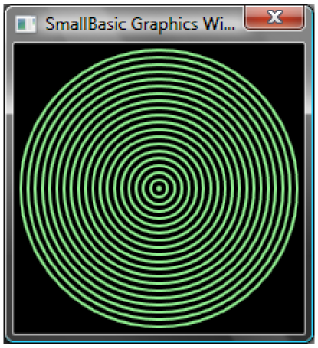


איור 33 - מלבנים

עיגולים

גרסה דומה של התוכנית הקודמת, מציירת עיגולים במקום מלבנים.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
  GraphicsWindow.DrawEllipse(100 - i, 100 - i, i * 2, i * 2)
EndFor
```



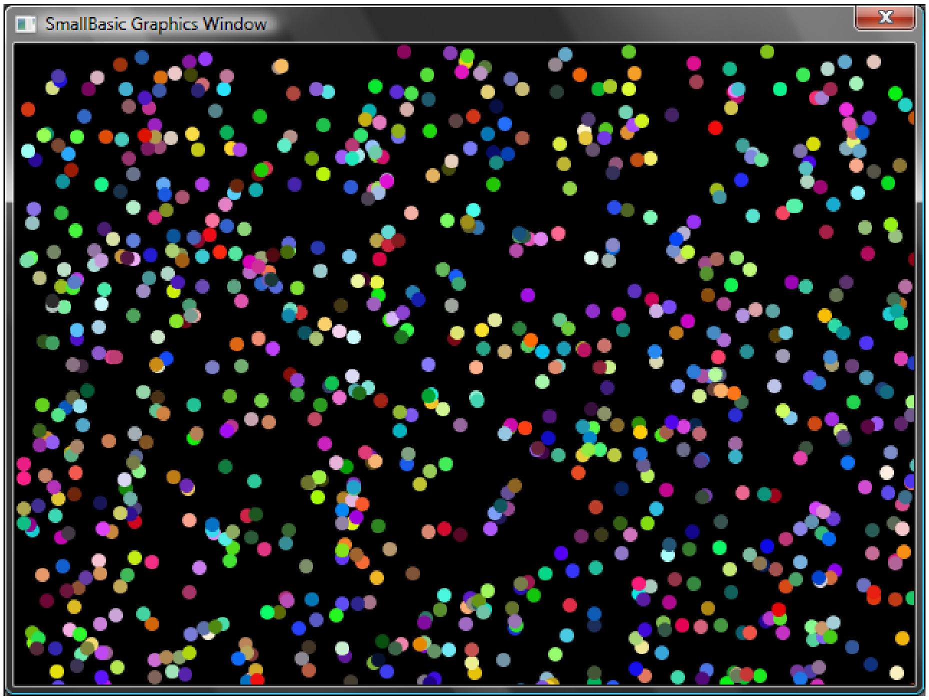
איור 34 - עיגולים

אקראיות

התוכנית הבאה משתמשת בפעולה GraphicsWindow.*GetRandomColor* כדי לקבוע צבע אקראי למברשת, ואז משתמשת בפעולה Math.*GetRandomNumber* כדי לקבוע את קואורדינאטות ה- *x* וה- *y* עבור עיגולים שהיא תצייר. בעזרת פעולות אלו המחזירות ערכים אקראיים, התוכנית נותנת תוצאות שונות בכל פעם שמריצים אותה.

```
GraphicsWindow.BackgroundColor = "Black"
```

```
For i = 1 To 1000
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
  x = Math.GetRandomNumber(640)
  y = Math.GetRandomNumber(480)
  GraphicsWindow.FillEllipse(x, y, 10, 10)
EndFor
```



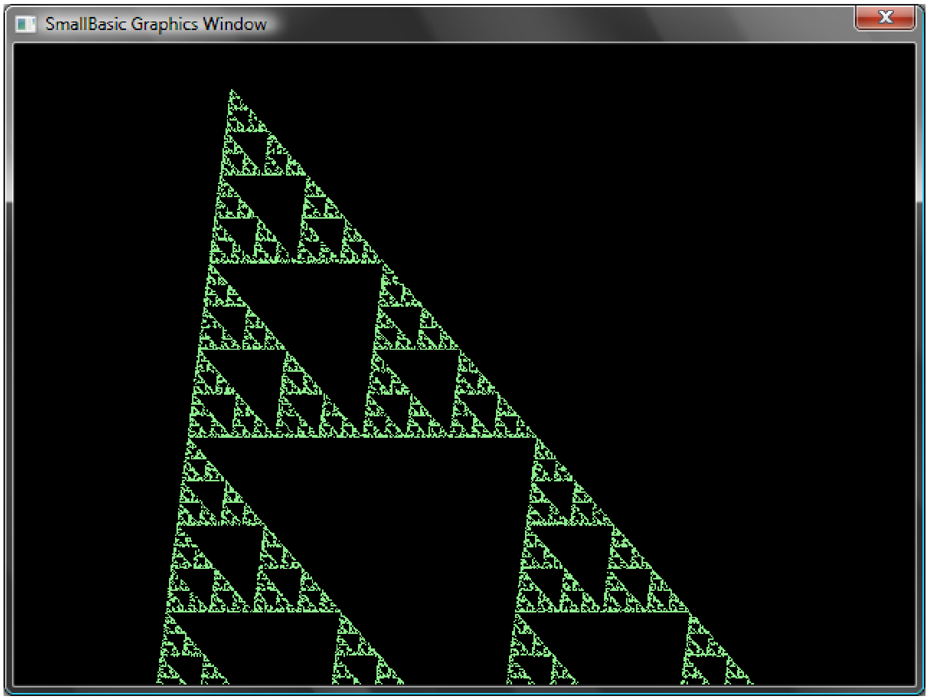
איור 35 - אקראיות

פרקטלים

התוכנית הבאה מציירת פרקטל משולשים פשוט על ידי שימוש במספרים אקראיים. *פרקטל* הוא צורה גיאומטרית הניתנת לחלוקה, כך שכל חלק דומה בצורתו לצורה המקורית. במקרה זה התוכנית מציירת מאות משולשים כאלו. מכיוון שהתוכנית רצה במשך מספר שניות, ניתן לראות איך נוצרים המשולשים מתוך נקודות פשוטות. הלוגיקה של התוכנית יותר קשה להסבר ולכן נשאיר את הבנתה כתרגיל עבורך.

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
  r = Math.GetRandomNumber(3)
  ux = 150
  uy = 30
```

```
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf

  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
EndFor
```



איור 36 – פרקטל משולשים

אם נרצה לראות את הנקודות ההולכות ויוצרות את הפרקטל, אפשר להכניס השהיה מכוונת באמצעות הפעולה **Program.***Delay*. פעולה זו מקבלת מספר המציין את אורך ההשהיה באלפיות שנייה. הנה שוב התוכנית עם השינוי בהדגשה.

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100
```

```
For i = 1 To 100000
  r = Math.GetRandomNumber(3)
  ux = 150
  uy = 30
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf

  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
  Program.Delay(2)
EndFor
```

הגדלת ההשהיה תאט יותר את התוכנית. שחקו עם המספר לקבלת התוצאה הרצויה לכם.

שינוי אחר שניתן לעשות לתוכנית הוא, החלפת השורה הבאה:

```
GraphicsWindow.SetPixel(x, y, "LightGreen")
```

בשורה הזאת:

```
color = GraphicsWindow.GetRandomColor()
GraphicsWindow.SetPixel(x, y, color)
```

שינוי זה יגרום לתוכנית לצייר את הפיקסלים (נקודות) בצבעים שונים.

פרק 8

גרפיקת צבים

לוגו

כבר משנות ה- 70 הייתה שפת תכנות פשוטה ועוצמתית בשם לוגו. השפה הייתה בעיקר בשימוש של חוקרים עד שנוספה לה "גרפיקת צבים". בשיטה זו ישנו "צב" על המסך המגיב לפקודות כמו "זוז קדימה", "פנה ימינה" וכדו'. באמצעות השימוש בצב אפשר היה ליצור ציורים מעניינים על המסך. מאפיין זה הפך את השפה לזמינה ומעניינת לאנשים מכל הגילאים וכך הפכה השפה לנפוצה מאד בשנות ה- 80.

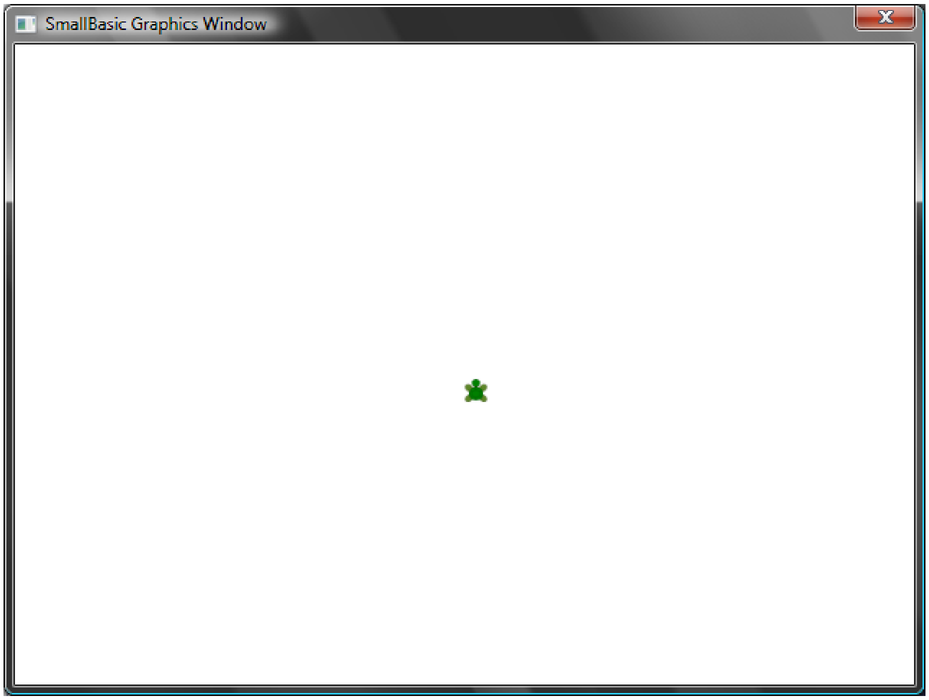
Small Basic מגיעה עם אובייקט **Turtle (צב)**, המגיב לפקודות רבות. בפרק זה נשתמש בצב ליצור גרפיקה על המסך.

הצב

כדי להתחיל, יש להציג את הצב על המסך וזאת נעשה בעזרת השורה הבאה:

```
Turtle.Show()
```

כשנריץ תוכנית זו נקבל חלון לבן ריק, כמו בפרק הקודם, רק שהפעם יופיע במרכז הצב שיקבל מאיתנו פקודות ציור.



איור 37 - רואים את הצב

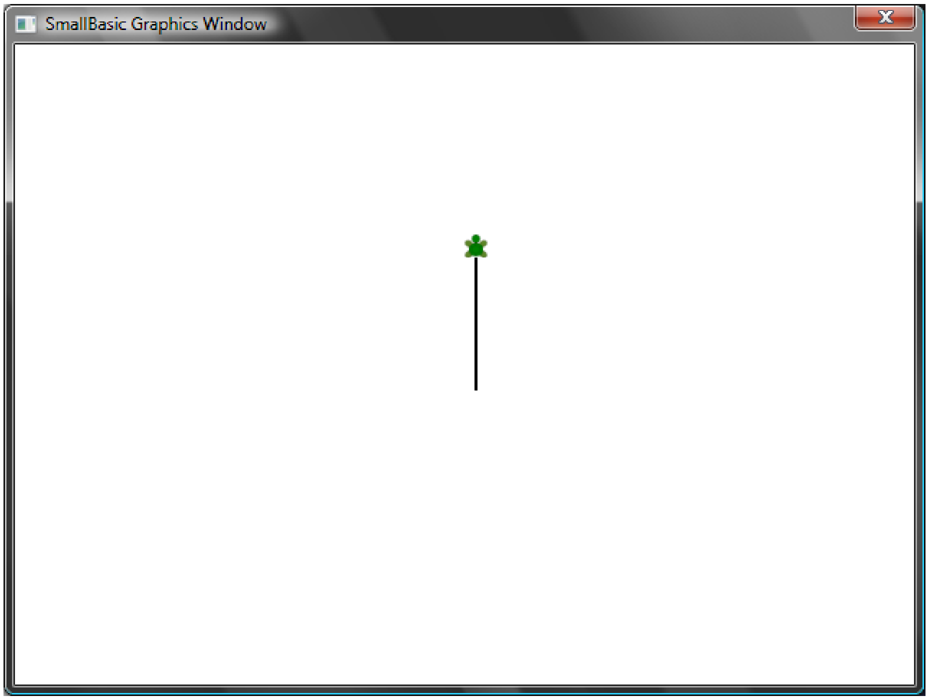
תזוזה וציור

אחת הפקודות, או הפעולות, שהצב מבין היא **Move (זוז)**. הפעולה מקבלת מספר כקלט. המספר מציין לצב כמה צעדים לזוז. למשל בדוגמה הבאה אנו מבקשים מהצב לזוז למרחק של 100 פיקסלים.

```
Turtle.Move(100)
```

כאשר נריץ תוכנית זו, נוכל לראות את הצב זז באיטיות למרחק 100 פיקסלים כלפי מעלה. תוך כדי שהוא זז נוצר קו מאחוריו. כאשר הצב מסיים לזוז, התוצאה תיראה כמו האיור שלמטה.

כאשר משתמשים בפעולות של הצב אין צורך לקרוא לפעולה ()Show. הצב יוצג באופן אוטומטי ברגע שמבצעים את אחת מהפעולות שלו.



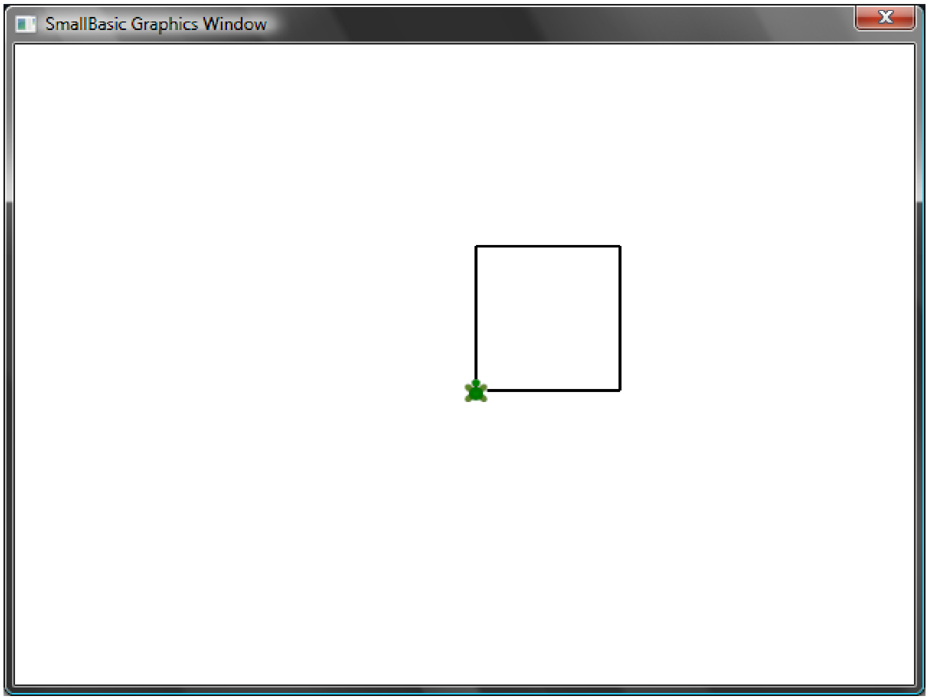
איור 38 - זוז מאה פיקסלים

ציור ריבוע

לריבוע יש ארבע צלעות, שתיים מאונכות ושתיים אופקיות. נוכל לצייר ריבוע אם נוכל לבקש שהצב יצייר קו, יפנה ימינה ויצייר קו נוסף וכך הלאה עד שכל הצדדים יושלמו. אם נתרגם זאת לתוכנית, היא תוכל להיראות כך:

```
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
```

כאשר נריץ תוכנית זו, נראה את הצב מצייר ריבוע, קו אחרי קו, כאשר התוצאה תיראה כך:



איור 39 - הצב מצייר ריבוע

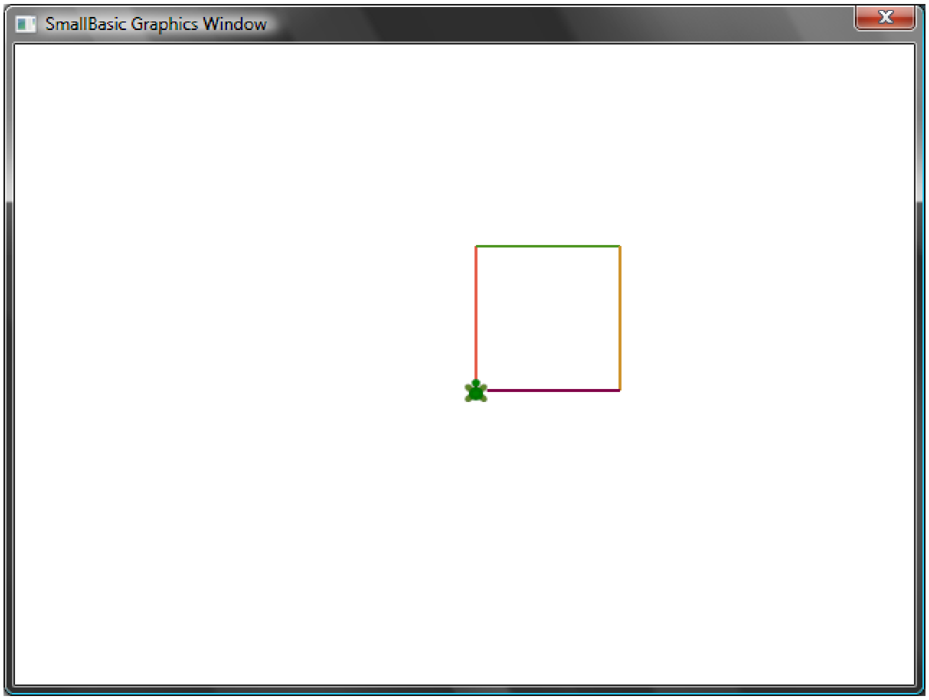
שימו לב שאנו משתמשים באותן פקודות שוב ושוב – ליתר דיוק ארבע פעמים, וכבר למדנו שאפשר להשתמש בלולאה כדי לבצע פעולות חוזרות. אם ניקח את התוכנית הקודמת ונשנה אותה כך שהיא תשתמש בלולאת **For..EndFor**, נקבל בהרצת התוכנית אותה התוצאה.

```
For i = 1 To 4
  Turtle.Move(100)
  Turtle.TurnRight()
EndFor
```

שינוי צבעים

הצב מצייר על אותו החלון הגראפי שראינו בפעם הקודמת. לכן כל הפעולות שכבר למדנו תקפות גם כאן. למשל התוכנית הבאה תצייר את הריבוע כך שכל צלע מצוירת בצבע אחר.

```
For i = 1 To 4
  GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
  Turtle.Move(100)
  Turtle.TurnRight()
EndFor
```



איור 40 - שינוי צבעים

ציור צורות מורכבות יותר

בנוסף לפעולות **TurnRight** ו- **TurnLeft**, יש לצב פעולת **Turn** (פנה). פעולה זו מקבלת קלט אחד המציין זווית לסיבוב. באמצעות פעולה זו אפשר לצייר כל מצולע (פוליגון). התוכנית הבאה מציירת משושה:

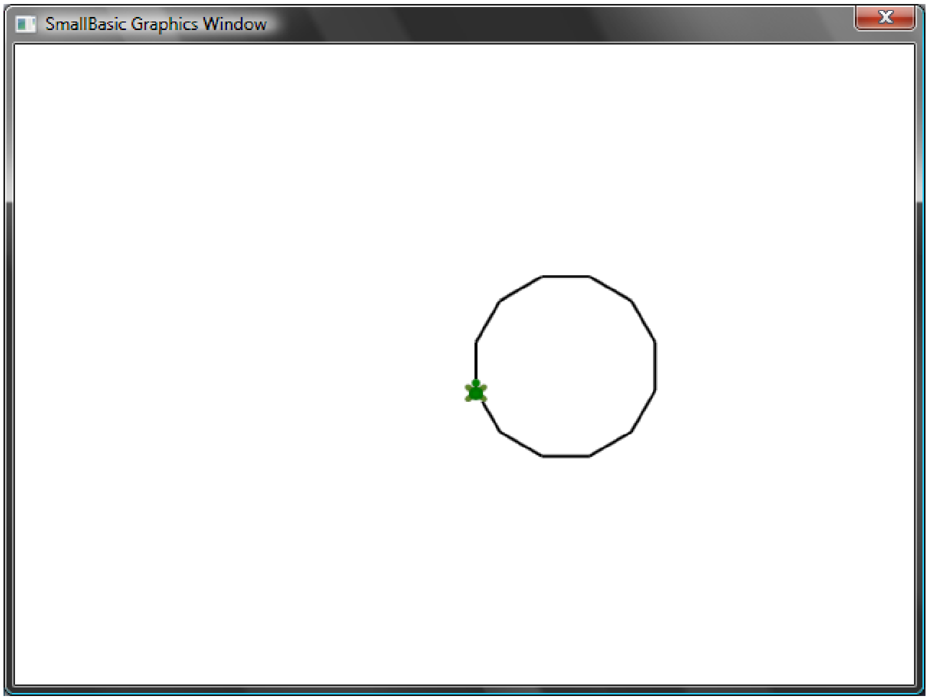
```
For i = 1 To 6
  Turtle.Move(100)
  Turtle.Turn(60)
EndFor
```

נסו את התוכנית כדי לאמת שהתוצאה היא משושה. שימו לב שמכיוון שהזווית בין הצלעות היא 60, השתמשנו ב- **Turn(60)**. באופן כללי, למצולע כזה שהוא שווה צלעות, אפשר לחשב בקלות את הזווית בין הצלעות על ידי חלוקת 360 במספר הצלעות. באמצעות ידע זה ושימוש במשתנים, נוכל לכתוב תוכנית כללית שיכולה לצייר כל מצולע שווה צלעות.

```
sides = 12

length = 400 / sides
angle = 360 / sides

For i = 1 To sides
  Turtle.Move(length)
```

```
  Turtle.Turn(angle)
EndFor
```

באמצעות תוכנית זו, אפשר לצייר מצולעים שונים על ידי עדכון המשתנה **sides**. השֹמה של 4 תיתן לנו את הריבוע שהתחלנו איתו. השֹמה של מספר גדול יותר למשל 50, תייצר מצולע שנראה כמעט כמו מעגל.



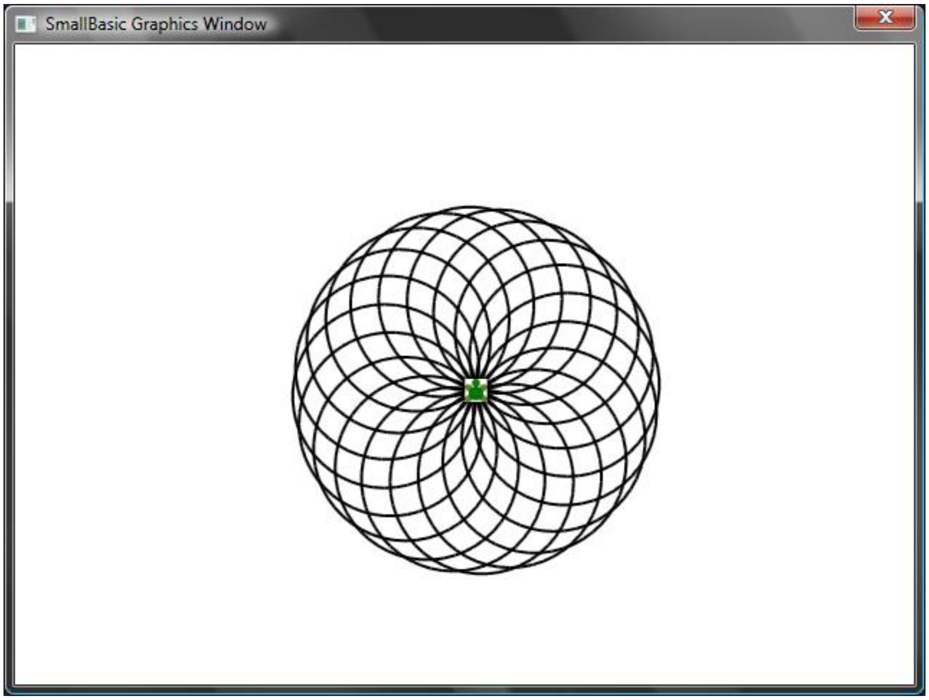
איור41 - ציור של מצולע בעל 12 צלעות

באמצעות הטכניקה שלמדנו נוכל גם לצייר מספר מעגלים, כל אחד בהזזה קטנה, כך שנקבל פלט מעניין.

```
sides = 50
length = 400 / sides
angle = 360 / sides

Turtle.Speed = 9

For j = 1 To 20
  For i = 1 To sides
    Turtle.Move(length)
    Turtle.Turn(angle)
  EndFor
  Turtle.Turn(18)
EndFor
```

בתוכנית שלמעלה גרמנו לצב לנוע מהר יותר על ידי קביעת המהירות ל- 9. אפשר לקבוע את המאפיין Speed לערך בין 1 ל- 10 ובכך לשלוט על מהירות ביצוע הציור.

בתוכנית שלמעלה ישנן שתי לולאות **For..EndFor** אחת בתוך השנייה. הלולאה הפנימית (**i** רץ מ- 1 עד **sides**) דומה לתוכנית של ציור המצולע והיא אחראית על ציור המעגל. הלולאה החיצונית (**j** רץ מ- 1 עד **20**) אחראית בכל ציור של אחד מעשרים מעגלים להזזה של הצב בזווית קטנה. סך כל הפעולות האלו יוצרות פלט בעל תבנית מעניינת, כמו שמופיע למטה.



איור 42 - מעגלים מוזזים

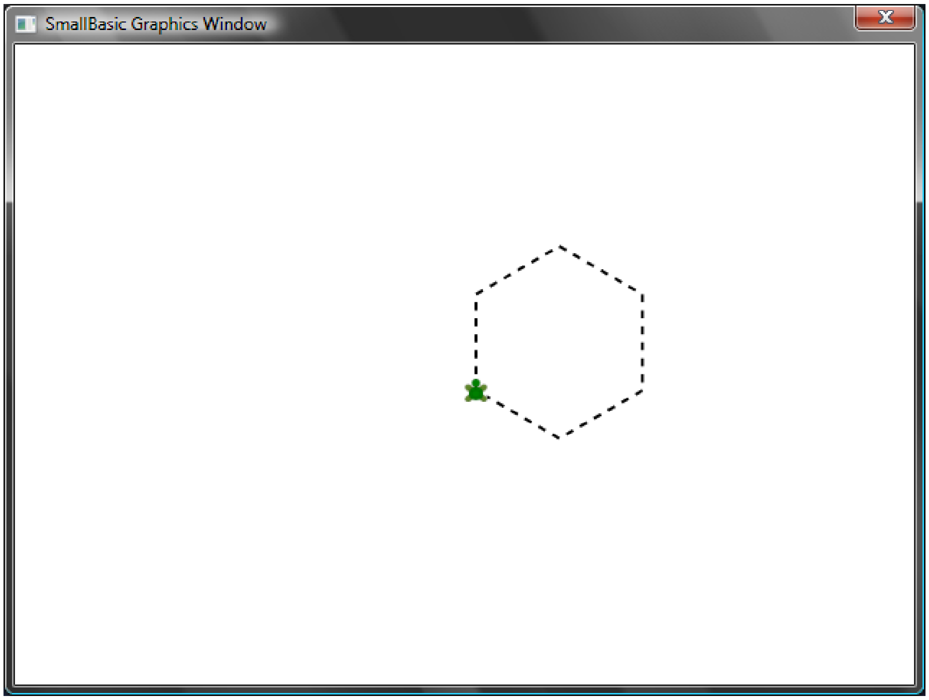
קפיצות

אפשר לגרום לצב לנוע בלי לצייר על ידי הפעולה **PenUp**. הדבר מאפשר להזיז את הצב לכל מקום במסך בלי להשאיר עקבות. לאחר שמגיעים למקום הרצוי אפשר לקרוא לפעולה **PenDown** שתחדש את הציור על ידי הצב. כך אפשר ליצור אפקטים שונים למשל ציור קווים מקווקווים. התוכנית הבאה מנצלת זאת לציור מצולע מקווקוו.

```
sides = 6

length = 400 / sides
angle = 360 / sides

For i = 1 To sides
  For j = 1 To 6
```

```
    Turtle.Move(length / 12)
    Turtle.PenUp()
    Turtle.Move(length / 12)
    Turtle.PenDown()
  EndFor
  Turtle.Turn(angle)
EndFor
```

גם בתוכנית זו יש שתי לולאות. הלולאה הפנימית אחראית על ציור קו מקווקו באמצעות תזוזות קטנות עם ציור ובלעדיו. הלולאה החיצונית אחראית על מספר הצלעות. בדוגמה זו השתמשנו בערך 6 עבור המשתנה **sides** ולכן התקבל משושה כמו שמופיע למטה.



איור 43 – שימוש ב- PenUp ו- PenDown

פרק 9

שגרות

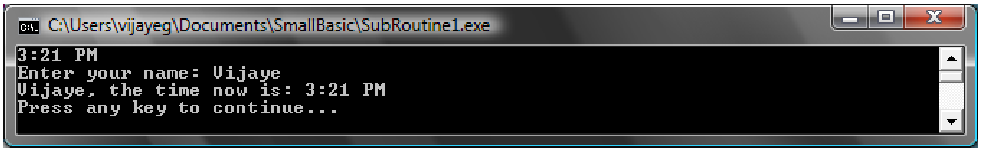
לעיתים קרובות כשכותבים תוכנית מחשב, נתקלים בצורך לבצע סט דומה של פעולות במקומות שונים בתוכנית. במקרים כאלו במקום לחזור על עצמינו שוב ושוב, נוכל להשתמש במנגנון *השגרה* (*Subroutine*, נקראת לפעמים גם פונקציה).

שגרה היא קטע מתוכנית גדולה יותר, קטע שאחראי בדרך כלל על פעולה ייחודית שאפשר להפעיל מכל מקום בתוכנית. שגרות מזוהות בתוכנית על ידי שם המופיע לאחר המילה השמורה **Sub**, ומסתיימות במילה השמורה **EndSub**. בדרך זו אנו מוסיפים לתוכנית פעולה חדשה. פעולה זו נוספת לפעולות שהמערכת תומכת בהן מראש. הדוגמה הבאה מכילה למשל שגרה ששמה *PrintTime* ותפקידה לכתוב לחלון הטקסט את הזמן הנוכחי.

```
Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
EndSub
```

עכשיו נציג תוכנית הכוללת שגרה זו וקוראת לה מספר פעמים במקומות שונים.

```
PrintTime()
TextWindow.Write("Enter your name: ")
name = TextWindow.Read()
TextWindow.Write(name + ", the time now is: ")
PrintTime()

Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
EndSub
```



איור 44 – קריאה לשגרה פשוטה

קוראים לשגרה על ידי שמה בצירוף סוגריים, בדומה לפעולות קיימות שכבר למדנו להשתמש בהם, הסוגריים ( ) הכרחיים לשם קריאה לפעולה.

יתרונות לשימוש בשגרות

כפי שראינו שגרות עוזרות להקטין את כמות הקוד שאנו כותבים. לאחר שכתבנו את השגרה *PrintTime*, אפשר לקרוא לה בכל מקום בתוכנית כך שיודפס הזמן הנוכחי.

שגרות יכולות גם לעזור לנו לפרק בעיה מורכבת לחלקים פשוטים יותר. נניח למשל שיש לנו משוואה מורכבת לפתור. נוכל לכתוב שגרות הפותרות חלקים קטנים של המשוואה ואז לחבר את התוצאות יחד כדי לקבל את הפתרון הדרוש.

שימו לב שב- Small Basic אפשר לקרוא לשגרה רק מאותה התוכנית. לא ניתן בצורה פשוטה לקרוא לשגרה בתוכנית אחרת.

שגרות גם עוזרות לשפר את יכולת הבנת התוכנית. במילים אחרות: אם נסמן חלקים מתוכנית עם שמות מתאימים, יהיה קל יותר לקרוא ולהבין אותה. נושא זה חשוב במיוחד כאשר אנו רוצים להבין תוכנית שמישהו אחר כתב, או אם אנו מעוניינים שאחרים יבינו את התוכנית שלנו. לעיתים, גם מי שכתב את התוכנית יכול להיעזר בכך לאחר שחולף זמן מכתיבתה.

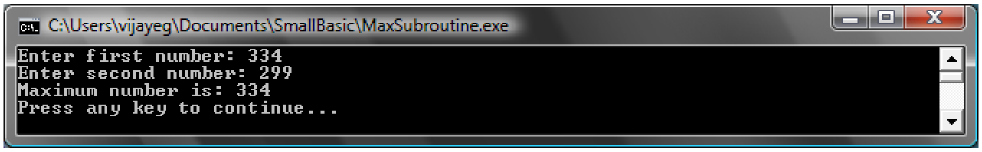
שימוש במשתנים

מתוך שגרה, אפשר לגשת ולהשתמש בכל משתנה של התוכנית. לדוגמה, התוכנית הבאה מקבלת שני מספרים וכותבת את הגדול מבין שניהם. שימו לב שהמשתנה *max* משמש גם בתוך השגרה וגם מחוצה לה.

```
TextWindow.Write("Enter first number: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Enter second number: ")
num2 = TextWindow.ReadNumber()

FindMax()
TextWindow.WriteLine("Maximum number is: " + max)

Sub FindMax
  If (num1 > num2) Then
    max = num1
  Else
```

```
    max = num2
  EndIf
EndSub
```

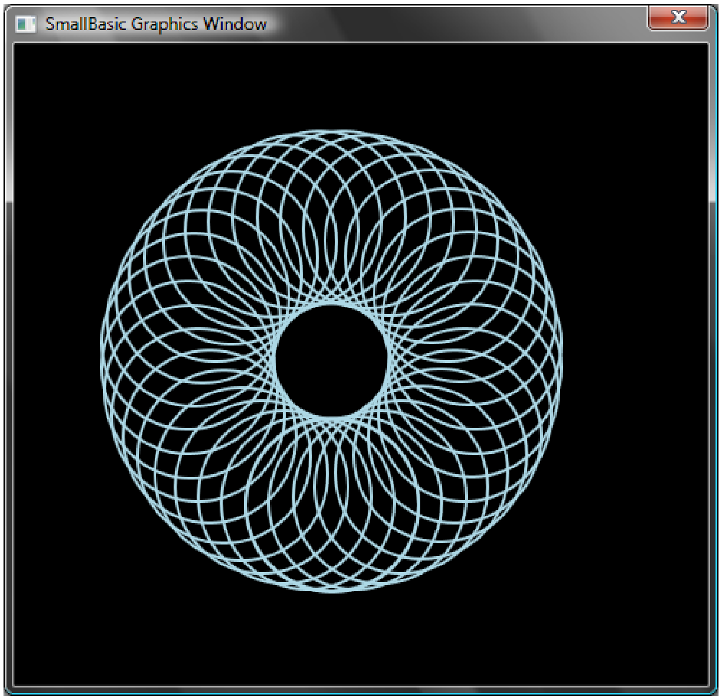
פלט התוכנית יכול להיראות כך:



איור 45 - הגדול בין שני מספרים באמצעות שגרה

בואו נראה תוכנית נוספת המדגימה את השימוש בשגרות. הפעם נכתוב תוכנית גראפית שתחשב נקודות שונות ותשמור אותם במשתנים בשם x ו- y. לאחר מכן התוכנית קוראת לשגרה **DrawCircleUsingCenter** שתפקידה לצייר מעגל שמרכזו נקבע לפי **x** ו- **y**.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 480
For i = 0 To 6.4 Step 0.17
  x = Math.Sin(i) * 100 + 200
  y = Math.Cos(i) * 100 + 200

  DrawCircleUsingCenter()
EndFor

Sub DrawCircleUsingCenter
  startX = x - 40
  startY = y - 40

  GraphicsWindow.DrawEllipse(startX, startY, 120, 120)
EndSub
```



איור 46 - דוגמה גראפית עם שגרות

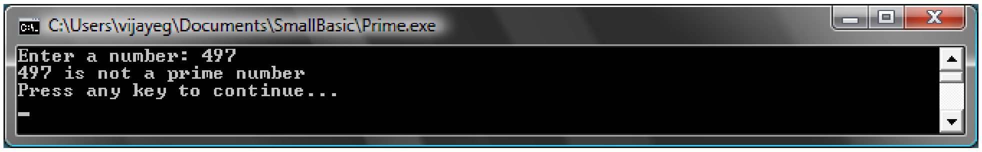
קריאה לשגרה מתוך לולאה

אפשר לקרוא לשגרה בתוך לולאה, כך שאותו סט של פקודות יתבצע שוב ושוב, אך ייתכן שבכל פעם הביצוע יהיה עם ערכים שונים במשתנים. לדוגמה, נניח שיש לנו שגרה בשם *PrimeCheck* הבודקת האם מספר הוא ראשוני (מספר המתחלק באחד ובעצמו בלבד). אפשר לכתוב תוכנית המבקשת מהמשתמש מספר ובאמצעות השגרה התוכנית בודקת האם המספר שהתקבל הוא ראשוני. התוכנית שלהלן מדגימה זאת:

```
TextWindow.Write("Enter a number: ")
i = TextWindow.ReadNumber()
isPrime = "True"
PrimeCheck()
If (isPrime = "True") Then
TextWindow.WriteLine(i + " is a prime number")
Else
TextWindow.WriteLine(i + " is not a prime number")
EndIf

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPrime = "False"
      Goto EndLoop
    EndIf
```

```
  Endfor
EndLoop:
EndSub
```

השגרה *PrimeCheck* לוקחת את המספר השמור ב- *i* ומנסה לחלק אותו במספרים קטנים יותר. אם נמצא מספר שמחלק את *i* ללא שארית, אז *i* אינו ראשוני. במקרה כזה השגרה מסמנת זאת על ידי הכנסת ערך "*False*" במשתנה *isPrime*, ויוצאת לתווית שבסוף השגרה על ידי קפיצה. אם השגרה לא מצאה מחלק כזה, *isPrime* נשאר עם ערכו ההתחלתי שהוא "*True*".

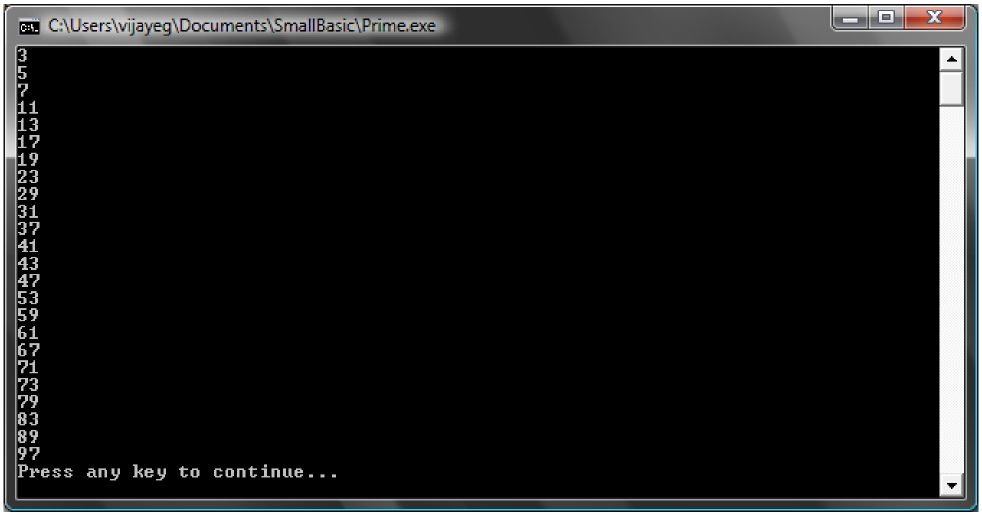


איור 47 - בדיקת ראשוניות

עכשיו כשפיתחנו את השגרה שבודקת ראשוניות של מספרים עבורנו, נוכל להשתמש בה כדי לרשום למשל את כל המספרים הראשונים עד 100 לדוגמה. קל למדי לשנות את התוכנית הקודמת כך שהקריאה ל- PrimeCheck תהיה מתוך לולאה. כך השגרה מקבלת ערך אחר לחישוב בכל ריצה של לולאה. נראה זאת בדוגמה הבאה:

```
For i = 3 To 100
  isPrime = "True"
  PrimeCheck()
  If (isPrime = "True") Then
  TextWindow.WriteLine(i)
  EndIf
EndFor

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPrime = "False"
      Goto EndLoop
    EndIf
  Endfor
EndLoop:
EndSub
```

בתוכנית זו, הערך של i מעודכן בכל ריצה של הלולאה, שבתוכה מתבצעת קריאה לשגרה PrimeCheck. השגרה לוקחת את ערכו של i ומחשבת האם הוא ראשוני. תוצאת החישוב נשמרת במשתנה isPrime שאליו ניגש הקוד בהמשך הלולאה. הערך של i נרשם אם התגלה שהוא ראשוני. מכיוון שהלולאה עוברת על הערכים מ- 3 ועד 100, אנו מקבלים רשימה של כל המספרים הראשוניים בטווח זה. הנה פלט התוכנית:



איור 48 - מספרים ראשוניים

פרק 10

מערכים

עכשיו שאתם כבר די בקיאים כנראה בשימוש במשתנים – אחרי הכול הגעתם עד לכאן ואתם עדיין נהנים, לא ככה?

אז בואו נבחן את התוכנית הראשונה שבה השתמשנו במשתנים:

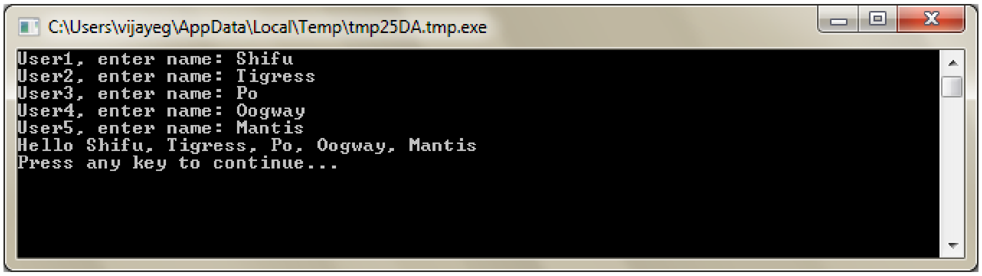
```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

בתוכנית זו קלטנו ושמרנו את שמו של המשתמש במשתנה שנקרא **name**. אחר כך אמרנו למשתמש "שלום". אם יש יותר ממשתמש אחד, נניח שיש חמישה כאלה למשל, כיצד נשמור את השמות של כולם? דרך אחת היא בדרך הזו:

```
TextWindow.Write("User1, enter name: ")
name1 = TextWindow.Read()
TextWindow.Write("User2, enter name: ")
name2 = TextWindow.Read()
TextWindow.Write("User3, enter name: ")
name3 = TextWindow.Read()
TextWindow.Write("User4, enter name: ")
name4 = TextWindow.Read()
TextWindow.Write("User5, enter name: ")
name5 = TextWindow.Read()

TextWindow.Write("Hello ")
```

```
TextWindow.Write(name1 + ", ")
TextWindow.Write(name2 + ", ")
TextWindow.Write(name3 + ", ")
TextWindow.Write(name4 + ", ")
TextWindow.WriteLine(name5)
```

אם נריץ תוכנית זו, נקבל את התוצאה הבאה:



איור 49 - בלי מערכים

די ברור שיש דרך יותר טובה לכתוב תוכנית פשוטה שכזו. במיוחד שתוכניות מחשב אמורות להיות טובות בביצוע משימות החוזרות על עצמן. אין סיבה שנצטרך לכתוב אותו קוד שוב ושוב עבור כל משתמש. נוכל לעשות זאת, אם תהיה דרך לשמור את השמות של כל המשתמשים במשתנה. אז נוכל להשתמש בלולאת ה- **For** שאותה אנחנו כבר מכירים. כאן יבואו לעזרתנו המערכים.

מהו מערך?

מערך הוא משתנה מסוג מיוחד שיכול לשמור יותר מערך אחד בו-זמנית. כך במקום ליצור חמישה משתנים לשמירת שמות המשתמש למשל **name1**, **name2**, **name3**, **name4** ו- **name5**, נוכל להשתמש במשתנה אחד בשם **name**. הדרך לשמור ערכים שונים בתוך משתנה כזה, היא להשתמש באינדקס או מציין. למשל **name[1]**, **name[2]**, **name[3]**, **name[4]** ו- **name[5]** יכולים לשמור כל אחד ערך שונה. המספרים 1, 2, 3, 4 ו- 5, נקראים מציינים של המערך.

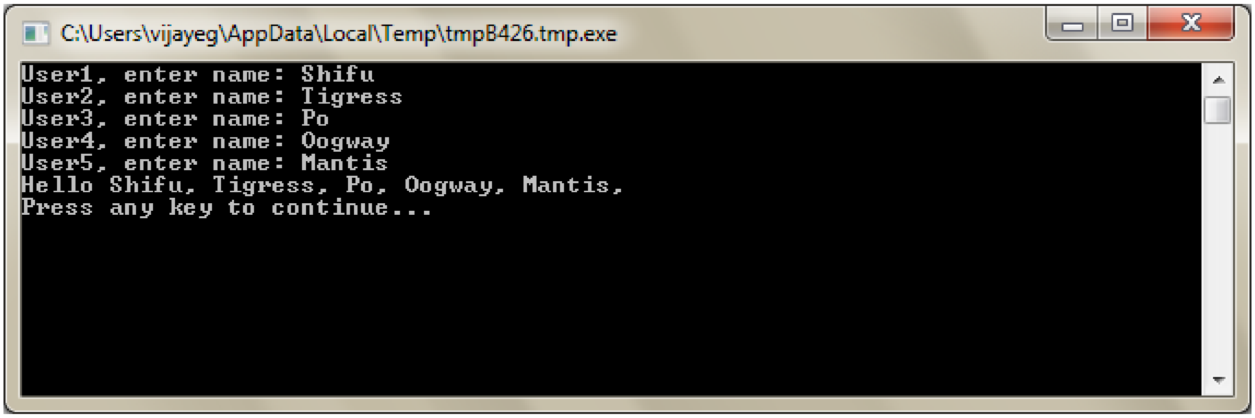
במבט ראשון **name[1]**, **name[2]**, **name[3]**, **name[4]** ו- **name[5]**, נראים משתנים שונים, אך למעשה יש כאן רק משתנה אחד. אולי תשאלו מה הרווחנו מכך? והתשובה היא שאת המציין אפשר לרשום גם כשם של משתנה או כל ביטוי אחר שמְחשב אותו. זה יאפשר לנו לגשת אל חלקי המערך השונים באמצעות לולאה.

בואו נראה כיצד לנצל את הידע החדש שזה עתה למדנו כדי לשפר את התוכנית הקודמת וזאת באמצעות מערכים.

```
For i = 1 To 5
  TextWindow.Write("User" + i + ", enter name: ")
  name[i] = TextWindow.Read()
EndFor

TextWindow.Write("Hello ")
```

```
For i = 1 To 5
  TextWindow.Write(name[i] + ", ")
EndFor
TextWindow.WriteLine("")
```

עכשיו התוכנית קצרה יותר ונוחה לקריאה. שימו לב לשתי השורות המובלטות. הראשונה שומרת ערך במערך והשנייה קוראת ערך מהמערך. הערך שהתוכנית שומרת ב- **name[1]** אינו מושפע ממה שנשמר ב- **name[2]**. לכן, אפשר להתייחס אליהם כמעט תמיד כשני משתנים שונים שיש להם שם משותף.



איור 50 - שימוש במערכים

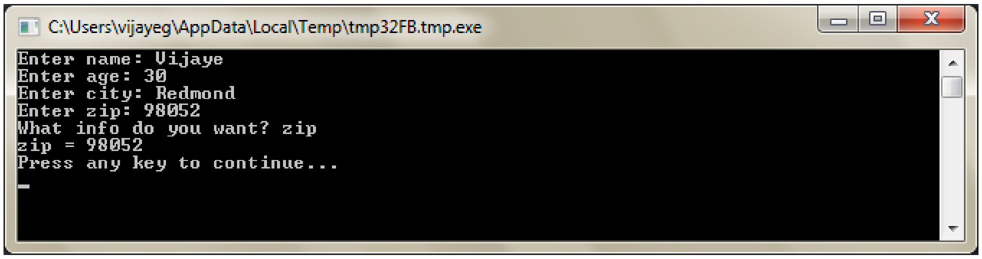
התוכנית המעודכנת מייצרת כמעט אותו פלט כמו התוכנית ללא מערכים. האם תוכלו למצוא את ההבדל בעצמכם? ...
ההבדל היחיד הוא הפסיק המופיע לאחר רשימת כל השמות. נוכל לתקן זאת על ידי שינוי לולאת ההדפסה:

```
TextWindow.Write("Hello ")
For i = 1 To 5
  TextWindow.Write(name[i])
  If i < 5 Then
    TextWindow.Write(", ")
  EndIf
EndFor
TextWindow.WriteLine("")
```

המציין של מערך

בתוכנית הקודמת השתמשנו במספרים כמציינים, זאת כדי לשמור וכדי להוציא ערכים מהמערך. מסתבר שהמציינים לא מוגבלים רק למספרים. אנחנו נראה שזה אפילו מאוד שימושי להשתמש במציינים טקסטואליים. לדוגמה, בתוכנית הבאה אנו מבקשים פריטי מידע אודות משתמש ולאחר מכן כותבים את המידע שהמשתמש מבקש.

```
TextWindow.Write("Enter name: ")
user["name"] = TextWindow.Read()
TextWindow.Write("Enter age: ")
user["age"] = TextWindow.Read()
TextWindow.Write("Enter city: ")
user["city"] = TextWindow.Read()
TextWindow.Write("Enter zip: ")
user["zip"] = TextWindow.Read()

TextWindow.Write("What info do you want? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + user[index])
```



איור 51- שימוש במציינים לא מספריים

יותר ממימד אחד

נניח שנרצה לשמור את השם ואת מספר הטלפון של כל החברים שלנו, ואחר כך נרצה לחפש מספר טלפון – מעין ספר טלפונים. כיצד נכתוב תוכנית כזו?

כמו בשמות משתנים, במצייני מערך אין תלות ברישיות של אותיות האנגלית (non-case sensitive).

במקרה כזה מעורבות שתי קבוצות של מציינים (הנקראות גם מימדים). נניח שנזהה כל חבר לפי הכינוי שלו, הכינוי יהפוך למציין הראשון במערך. ברגע שנשתמש במציין הראשון כדי להגיע למקום בו מאוחסנים פרטי החבר, נוכל להשתמש במצייני שם (**name**) ומספר טלפון (**phone**) כדי לקבל את המידע הדרוש.

שמירת המידע תתבצע בדרך הבאה:

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"
```

```
friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"
```

מכיוון שלמערך **friends** יש שני מציינים הוא נקרא מערך דו-מימדי (בעל שני מימדים).

ברגע שהמידע מאוחסן, נוכל לבקש את הכינוי של החבר כקלט ולרשום את המידע ששמרנו עליו. הנה התוכנית המלאה שעושה זאת:

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"

TextWindow.Write("Enter the nickname: ")
nickname = TextWindow.Read()

TextWindow.WriteLine("Name: " + friends[nickname]["Name"])
TextWindow.WriteLine("Phone: " + friends[nickname]["Phone"])
```

איור 52 - ספר טלפונים פשוט

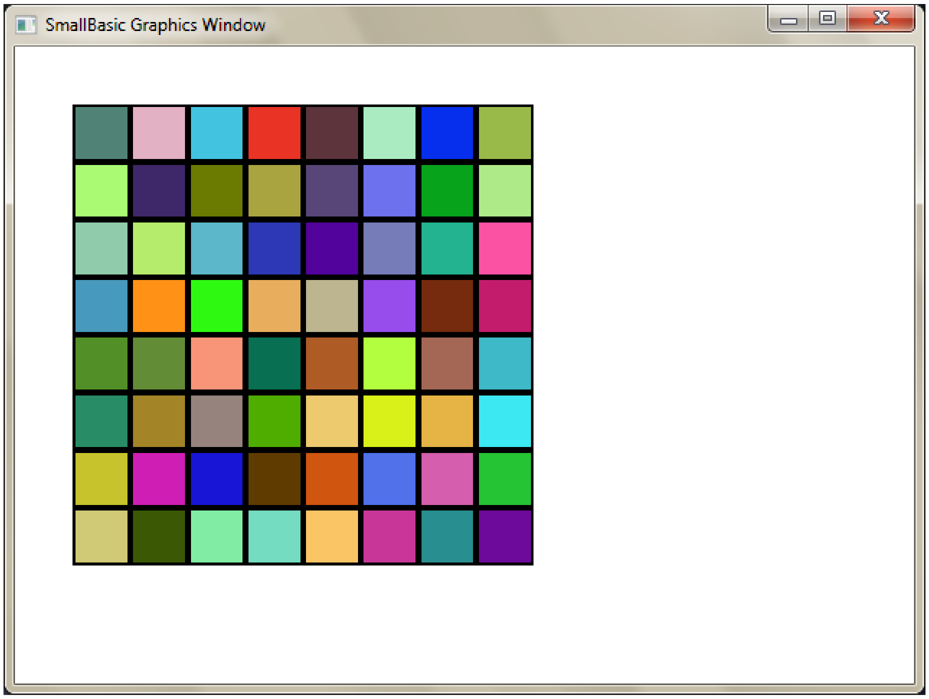
שימוש במערכים לייצוג רשתות

שימוש נפוץ במערכים הוא ייצוג בזיכרון של רשתות או טבלאות. לרשתות יש שורות ועמודות שמתאימות יפה למערך דו-מימדי. הנה תוכנית פשוטה שמסדרת ריבועים ברשת:

```
rows = 8
columns = 8
size = 40

For r = 1 To rows
```

```
  For c = 1 To columns
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    boxes[r][c] = Shapes.AddRectangle(size, size)
    Shapes.Move(boxes[r][c], c * size, r * size)
  EndFor
EndFor
```

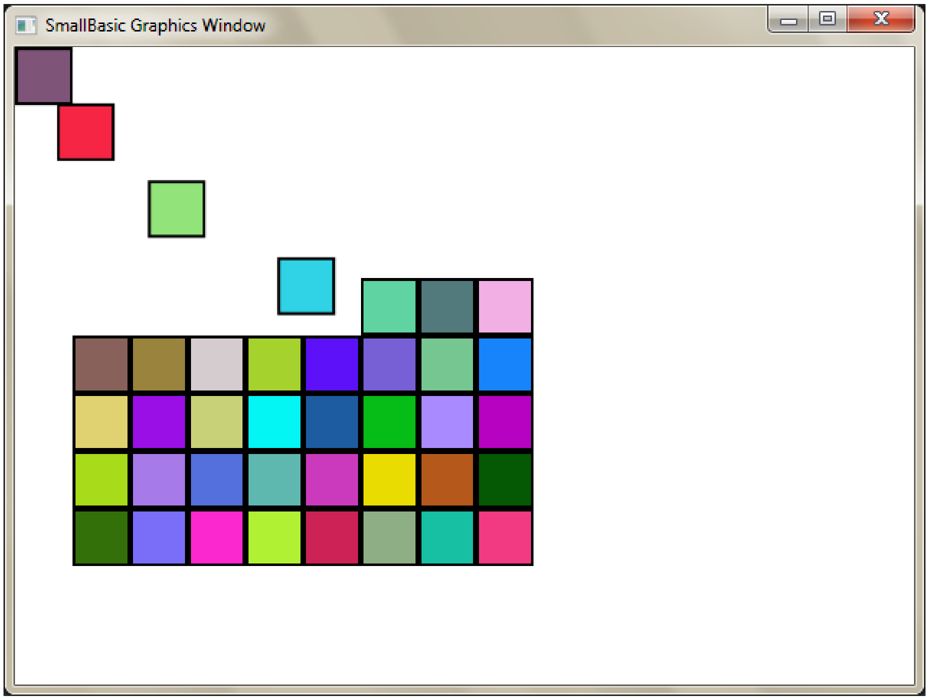
תוכנית זו מוסיפה למסך מלבנים ומסדרת אותם ברשת 8 על 8. בנוסף לסידור הצורות התוכנית שומרת אותם במערך, וכך אפשר לגשת ולהשתמש בהם שוב.



איור 53 - סידור ריבועים ברשת

לדוגמה, הוספת הקוד הבא בסוף התוכנית הקודמת, תייצר אנימציה שבה הריבועים עוברים לפינה השמאלית העליונה:

```
For r = 1 To rows
  For c = 1 To columns
    Shapes.Animate(boxes[r][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```



איור 54 - שימוש בריבועים שברשת

פרק 11

אירועים ואינטראקטיביות

בשני הפרקים הראשונים למדנו שלאובייקטים יש מאפיינים ופעולות. בנוסף לאלו, ישנם אובייקטים שיש להם גם *אירועים* (***Events***). אירועים הם כמו סימונים שמתקבלים מהמחשב, זאת בתגובה לפעולות של המשתמש. פעולות כאלו יכולות להיות הזזת העכבר או לחיצה עליו. במובן מסוים אירועים הם ההפך מפעולות. במקרה של פעולה, המתכנתים קוראים לה כדי שיתבצע משהו במחשב. לעומת זאת באירועים, התוכנית מודיעה שהתרחש משהו מעניין שעבורו המתכנתים מכינים תגובה מתאימה.

מתי מתאים להשתמש באירועים?

אירועים הם מרכזיים עבור הכנסת אינטראקטיביות לתוכנית. הם יכולים לאפשר תקשורת של המשתמש עם התוכנית. נניח שברצוננו לכתוב תוכנית איקס-עיגול. ברור שנרצה לאפשר למשתמש לבחור את הצעד שלו, נכון? כאן באים אירועים לעזרתנו בקבלת הקלט מהמשתמש. זה אולי נראה קשה לתפיסה בהתחלה, אך אל דאגה מייד נראה דוגמה שתעזור לנו להבין מהם אירועים וכיצד להשתמש בהם.

הנה תוכנית פשוטה שבה יש משפט אחד ועוד שגרה אחת. השגרה משתמשת בפעולה *ShowMessage* של החלון הגראפי בכדי להציג תיבת הודעה (*message box*) למשתמש.

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  GraphicsWindow.ShowMessage("You Clicked.", "Hello")
EndSub
```

שימו לב לשורה הראשונה בתוכנית. בשורה זו אנו מבצעים השמה של שם השגרה לאירוע **MouseDown** (העכבר נלחץ) של החלון הגראפי. שימו לב שהשימוש ב- MouseDown דומה לדרך השימוש במאפיינים של אובייקטים, אלא שבמקום השמה של ערך רגיל, אנחנו משתמשים בשם של השגרה *OnMouseDown*. מה שמיוחד באירועים הוא שהם גורמים לכך שבכל פעם שמתרחש האירוע, השגרה מתבצעת. במקרה שלנו, השגרה OnMouseDown

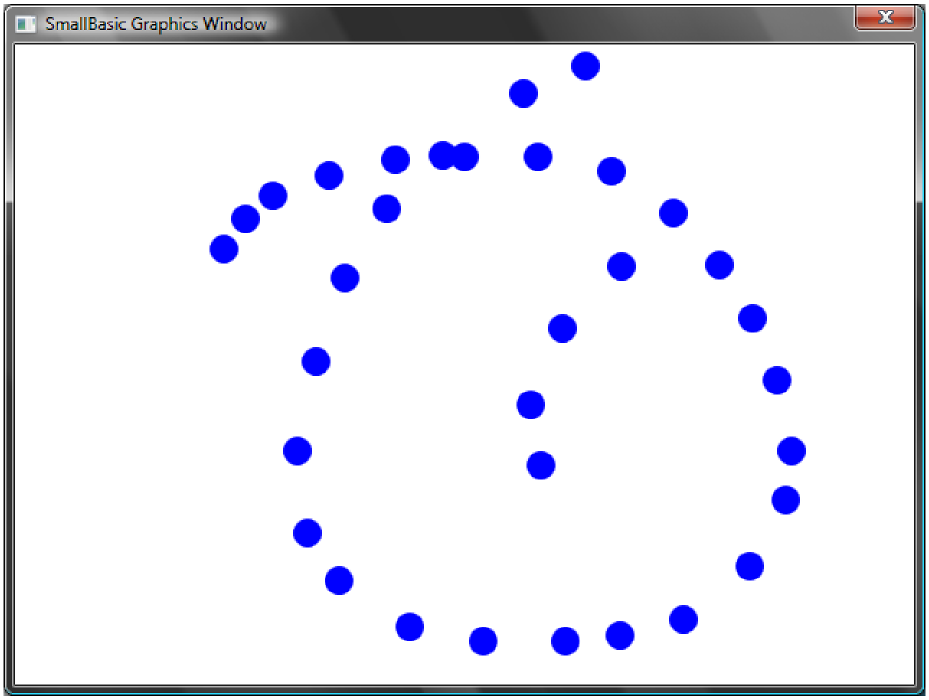
מתבצעת בכל פעם שהמשתמש לוחץ עם העכבר על החלון הגראפי. אתם מוזמנים לנסות בעצמכם, על ידי הרצת התוכנית. בכל פעם שתלחצו על החלון הגראפי עם העכבר, תראו תיבת הודעה כמו בתמונה שלמטה.

איור 55- תגובה לאירוע

שיטה זו של טיפול באירועים מאפשרת תוכניות מעניינות ויצירתיות. לתוכניות כאלו קוראים גם תוכניות מונחות אירועים.

תוכלו גם לשנות את השגרה OnMouseDown כך שתבצע דברים שונים מאשר להראות תיבת שיחה. למשל בתוכנית הבאה נצייר נקודות כחולות גדולות במקומות שבהם המשתמש לוחץ:

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
  GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```



איור 56 - טיפול בלחיצות על העכבר

שימו לב שבתוכנית זו השתמשנו במאפיינים *MouseX* ו- *MouseY* כדי לקבל את הקואורדינאטות של מיקום העכבר. השתמשנו בקואורדינאטות אלו כמרכזי העיגולים לציור.

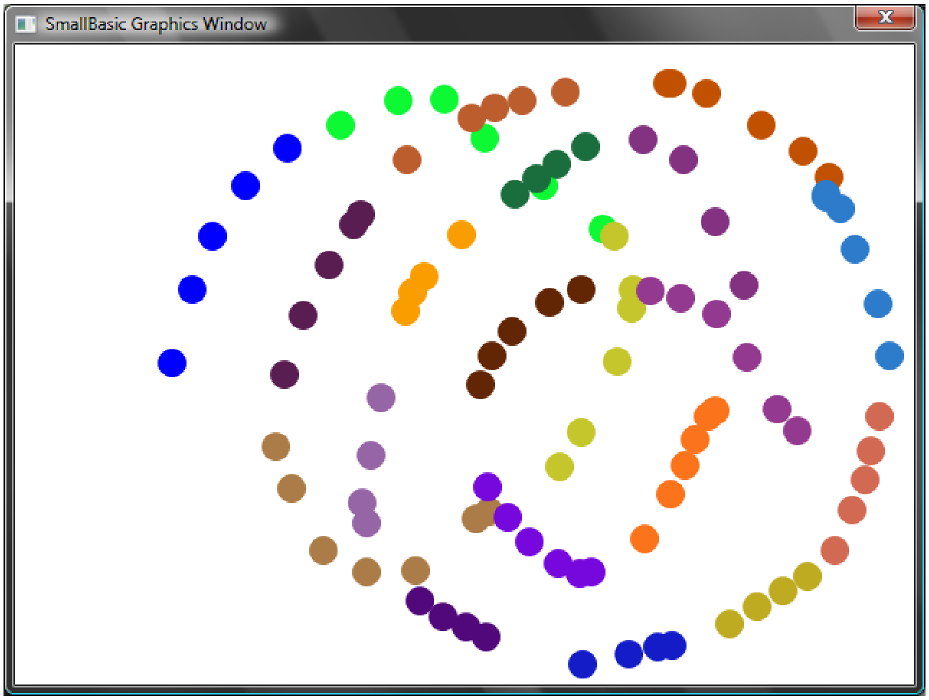
טיפול במספר אירועים

אין הגבלה למספר האירועים שבהם התוכנית יכולה לטפל בהם. אפשר גם ששגרה אחת תטפל במספר אירועים. יחד עם זאת בכל אירוע אפשר לטפל רק פעם אחת על ידי שגרה אחת. אם נבצע השמה של שתי שגרות לאירוע מסוים, רק השגרה השנייה תיקרא.

כדי להדגים זאת, בואו נוסיף לדוגמה הקודמת שגרה שמטפלת בלחיצות על מקשים. נדאג ששגרה זו תשנה את צבע המברשת בלחיצה על מקש, כך שכאשר העכבר יילחץ נקבל בכל פעם נקודה בצבע חדש:

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown
GraphicsWindow.KeyDown = OnKeyDown

Sub OnKeyDown
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
EndSub

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
  GraphicsWindow.FillEllipse(x, y, 20, 20)
```

```
EndSub
```



איור57- טיפול במספר אירועים

כשנריץ תוכנית זו, נקבל בלחיצה הראשונה על העכבר נקודה כחולה. עכשיו כשנלחץ על מקש כלשהו במקלדת, השגרה *OnKeyDown* תתבצע, דבר שישנה את צבע המברשת. מעכשיו הנקודות שיתקבלו בלחיצה על העכבר יהיו בצבע החדש שנבחר וכך נוכל לקבל נקודות צבעוניות.

תוכניות ציור

מצוידים בשגרות ואירועים, נוכל לכתוב תוכנית המאפשרת למשתמשים לצייר על החלון. אולי תופתעו לראות כמה קל לכתוב תוכנית כזו, אם מחלקים אותה לחלקים קטנים. בשלב ראשון בואו נכתוב תוכנית שתאפשר למשתמש להזיז את סמן העכבר לכל מקום בחלון, ותוך כדי כך להשאיר עקבות בכל מקום שמזיזים אליו את העכבר.

```
GraphicsWindow.MouseMove = OnMouseMove

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  GraphicsWindow.DrawLine(prevX, prevY, x, y)
  prevX = x
  prevY = y
EndSub
```

כשנריץ את התוכנית, נראה להפתעתנו שהקו הראשון תמיד מצויר מהפינה השמאלית העליונה (0, 0). אפשר לתקן זאת על ידי הוספת טיפול גם באירוע *MouseDown* ושם כבר לשמור את הערכים של *prevX* ו- *prevY*.

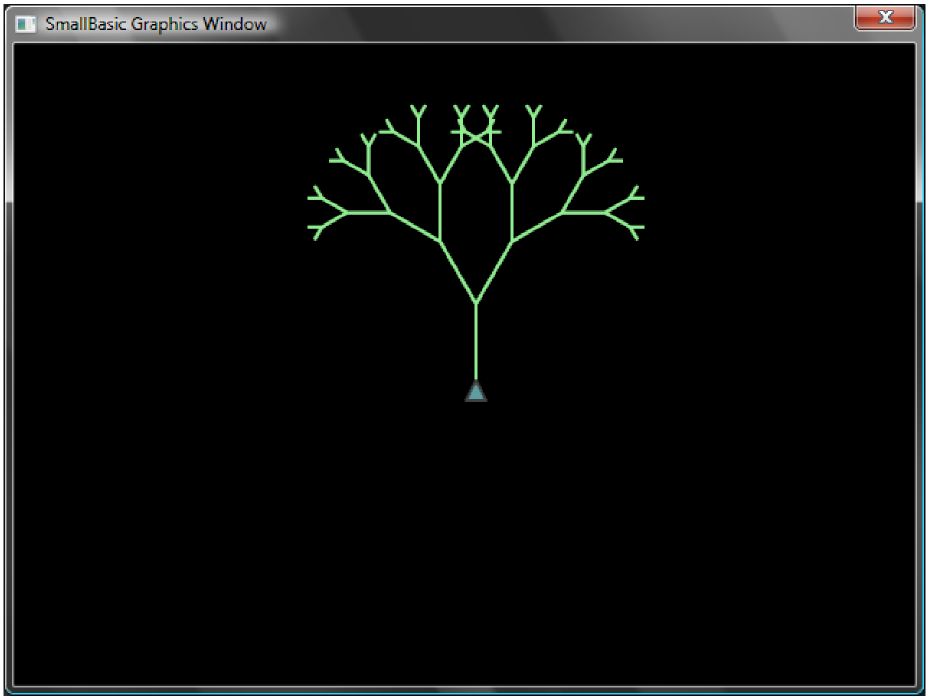
בנוסף, נרצה שהעקבות יצוירו רק כאשר כפתור העכבר לחוץ. בתזוזה אחרת לא נרצה שהקו יצויר. אפשר לממש התנהגות זו על ידי שימוש במאפיין *IsLeftButtonDown* של אובייקט ה- ***Mouse*** (עכבר). מאפיין זה אומר לנו האם כרגע כפתור העכבר השמאלי לחוץ. כאשר ערך זה אמת, נצייר קו ואחרת, לא נצייר אותו.

```
GraphicsWindow.MouseMove = OnMouseMove
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  prevX = GraphicsWindow.MouseX
  prevY = GraphicsWindow.MouseY
EndSub

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  If (Mouse.IsLeftButtonDown) Then
    GraphicsWindow.DrawLine(prevX, prevY, x, y)
  EndIf
  prevX = x
  prevY = y
EndSub
```

נספח א

דוגמאות חביבות

פרקטל עם הצב



איור 58 - הצב מצייר פרקטל עץ

```
angle = 30
delta = 10
distance = 60
Turtle.Speed = 9
GraphicsWindow.BackgroundColor = "Black"
```

```
GraphicsWindow.PenColor = "LightGreen"
DrawTree()

Sub DrawTree
  If (distance > 0) Then
    Turtle.Move(distance)
    Turtle.Turn(angle)

Stack.PushValue("distance", distance)
    distance = distance - delta
DrawTree()
    Turtle.Turn(-angle * 2)
    DrawTree()
    Turtle.Turn(angle)
    distance = Stack.PopValue("distance")

    Turtle.Move(-distance)
  EndIf
EndSub
```

תמונות משירות התמונות Flickr

איור 59 - אחזור תמונות מ- Flickr

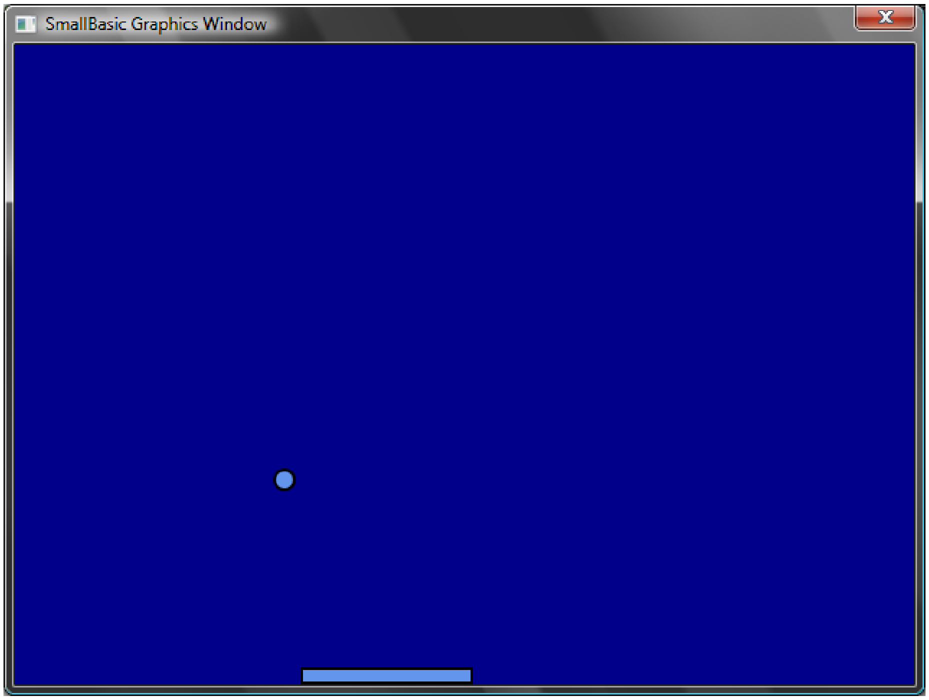
```
GraphicsWindow.BackgroundColor = "Black"
```

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
    pic = Flickr.GetRandomPicture("mountains, river")
    GraphicsWindow.DrawResizedImage(pic, 0, 0, 640, 480)
EndSub
```

טפט דינמי לשולחן העבודה

```
For i = 1 To 10
  pic = Flickr.GetRandomPicture("mountains")
  Desktop.SetWallPaper(pic)
  Program.Delay(10000)
EndFor
```

משחק מחבט



איור60- משחק מחבט

```
GraphicsWindow.BackgroundColor = "DarkBlue"
paddle = Shapes.AddRectangle(120, 12)
ball = Shapes.AddEllipse(16, 16)
GraphicsWindow.MouseMove = OnMouseMove
```

```
x = 0
y = 0
deltaX = 1
deltaY = 1

RunLoop:
  x = x + deltaX
  y = y + deltaY

  gw = GraphicsWindow.Width
  gh = GraphicsWindow.Height
  If (x >= gw - 16 or x <= 0) Then
    deltaX = -deltaX
  EndIf
  If (y <= 0) Then
    deltaY = -deltaY
  EndIf

  padX = Shapes.GetLeft(paddle)
  If (y = gh - 28 and x >= padX and x <= padX + 120) Then
    deltaY = -deltaY
  EndIf

Shapes.Move(ball, x, y)
  Program.Delay(5)

  If (y < gh) Then
    Goto RunLoop
  EndIf

GraphicsWindow.ShowMessage("You Lose", "Paddle")

Sub OnMouseMove
  paddleX = GraphicsWindow.MouseX
Shapes.Move(paddle, paddleX - 60, GraphicsWindow.Height - 12)
EndSub
```

נספח ב

צבעים

הנה רשימת שמות צבעים הנתמכים על ידי Small Basic, מחולקים לפי גוונים בסיסיים.

צבעים אדומים

IndianRed	#CD5C5C
LightCoral	#F08080
Salmon	#FA8072
DarkSalmon	#E9967A
LightSalmon	#FFA07A
Crimson	#DC143C
Red	#FF0000
FireBrick	#B22222
DarkRed	#8B0000

צבעים ורודים

Pink	#FFC0CB
LightPink	#FFB6C1
HotPink	#FF69B4
DeepPink	#FF1493
MediumVioletRed	#C71585
PaleVioletRed	#DB7093

צבעים כתומים

LightSalmon	#FFA07A
Coral	#FF7F50
Tomato	#FF6347
OrangeRed	#FF4500
DarkOrange	#FF8C00
Orange	#FFA500

צבעים צהובים

Gold	#FFD700
Yellow	#FFFF00
LightYellow	#FFFFE0
LemonChiffon	#FFFACD
LightGoldenrodYellow	#FAFAD2
PapayaWhip	#FFEFD5
Moccasin	#FFE4B5
PeachPuff	#FFDAB9
PaleGoldenrod	#EEE8AA

Khaki	#F0E68C
DarkKhaki	#BDB76B

צבעים סגולים

Lavender	#E6E6FA
Thistle	#D8BFD8
Plum	#DDA0DD
Violet	#EE82EE
Orchid	#DA70D6
Fuchsia	#FF00FF
Magenta	#FF00FF
MediumOrchid	#BA55D3
MediumPurple	#9370DB
BlueViolet	#8A2BE2
DarkViolet	#9400D3
DarkOrchid	#9932CC
DarkMagenta	#8B008B
Purple	#800080
Indigo	#4B0082
SlateBlue	#6A5ACD
DarkSlateBlue	#483D8B
MediumSlateBlue	#7B68EE

צבעים ירוקים

GreenYellow	#ADFF2F
Chartreuse	#7FFF00
LawnGreen	#7CFC00
Lime	#00FF00
LimeGreen	#32CD32
PaleGreen	#98FB98
LightGreen	#90EE90
MediumSpringGreen	#00FA9A
SpringGreen	#00FF7F
MediumSeaGreen	#3CB371
SeaGreen	#2E8B57
ForestGreen	#228B22
Green	#008000
DarkGreen	#006400
YellowGreen	#9ACD32
OliveDrab	#6B8E23
Olive	#808000
DarkOliveGreen	#556B2F
MediumAquamarine	#66CDAA
DarkSeaGreen	#8FBC8F
LightSeaGreen	#20B2AA
DarkCyan	#008B8B
Teal	#008080

צבעים כחולים

Aqua	#00FFFF
Cyan	#00FFFF
LightCyan	#E0FFFF
PaleTurquoise	#AFEEEE
Aquamarine	#7FFFD4
Turquoise	#40E0D0
MediumTurquoise	#48D1CC
DarkTurquoise	#00CED1
CadetBlue	#5F9EA0
SteelBlue	#4682B4
LightSteelBlue	#B0C4DE
PowderBlue	#B0E0E6
LightBlue	#ADD8E6
SkyBlue	#87CEEB
LightSkyBlue	#87CEFA

DeepSkyBlue	#00BFFF
DodgerBlue	#1E90FF
CornflowerBlue	#6495ED
MediumSlateBlue	#7B68EE
RoyalBlue	#4169E1
Blue	#0000FF
MediumBlue	#0000CD
DarkBlue	#00008B
Navy	#000080
MidnightBlue	#191970

צבעים חומים

Cornsilk	#FFF8DC
BlanchedAlmond	#FFEBCD
Bisque	#FFE4C4
NavajoWhite	#FFDEAD
Wheat	#F5DEB3
BurlyWood	#DEB887
Tan	#D2B48C
RosyBrown	#BC8F8F
SandyBrown	#F4A460
Goldenrod	#DAA520
DarkGoldenrod	#B8860B
Peru	#CD853F
Chocolate	#D2691E
SaddleBrown	#8B4513
Sienna	#A0522D
Brown	#A52A2A
Maroon	#800000

צבעים לבנים

White	#FFFFFF
Snow	#FFFAFA
Honeydew	#F0FFF0
MintCream	#F5FFFA
Azure	#F0FFFF
AliceBlue	#F0F8FF
GhostWhite	#F8F8FF
WhiteSmoke	#F5F5F5
Seashell	#FFF5EE
Beige	#F5F5DC
OldLace	#FDF5E6
FloralWhite	#FFFAF0
Ivory	#FFFFF0
AntiqueWhite	#FAEBD7
Linen	#FAF0E6
LavenderBlush	#FFF0F5
MistyRose	#FFE4E1

צבעים אפורים

Gainsboro	#DCDCDC
LightGray	#D3D3D3
Silver	#C0C0C0
DarkGray	#A9A9A9
Gray	#808080
DimGray	#696969
LightSlateGray	#778899
SlateGray	#708090
DarkSlateGray	#2F4F4F
Black	#000000