**שפת C#**

**דרישת קדם**

WPF היא תשתית של מיקרוסופט, על מנת לעבוד ב-WPF צריך להכיר את שפת התכנות C#.  
שפת C# היא שפה מונחת אובייקטים נפוצה מאוד בפיתוח על מערכות של מיקרוסופט.  
מכיוון שאתם כבר מכירים שפות תכנות כמו C, C++, Python המעבר לתכנות ב- C# יהיה קל מאוד, כל מה שאתם צריכים להכיר זה את התחביר של השפה.

על C# תוכלו ללמוד מהמסמך המצורף "C# - רקע ותחביר". המסמך יכול לשמש אתכם בתור Reference לאורך כל הלמידה, כך שלא תצטרכו ללמוד את כל התחביר עכשיו.

**קצת על C#**

נכון להיות אתם מכירים שתי שפות תכנות: C/C++ ו-Python.

בתרגיל מעבדה זה נכיר שפה שלישית הנקראת C# שפותחה בתחילת שנות ה- 2000 והיא שונה משתי האחרות.

C# היא "שפה מנוהלת", המשמעות של המונח היא שהקוד של התוכנה יבוצע תחת ניהול של מכונה וירטואלית, תהליך שרץ ברקע, מנהל את הקוד ומספק לו שירותים רבים (כמו ניהול זכרון למשל).

ב- C++ וב- Python ניתן לפתח מהמסד מבלי להישען ומבלי להיסתך על אף ספריה חיצונית.  
בניגוד אליהן, C# אינה עומדת בפני עצמה אלא היא שפת תכנות שמתבססת ונשענת על תשתית תוכנה (פלטפורמה) הנקראת .NET Framework (או המקבילה בקוד פתוח הנקראת .NET Core וניתנת להרצה על Linux ועל IOS).

הפלטפורמה מספקת לתוכנות שרצות מעליה כלים ושירותים רבים כגון ניהול זיכרון, שליטה בהתקני קלט/פלט, ממשק משתמש גרפי, תקשורת, אבטחה, אסינכרוניות, עבודה מול מסדי נתונים, כתיבה בצד שרת ועוד שירותים רבים אחרים.  
הפלטפורמה היא מאוד גדולה ומקיפה כמעט את כל תחומי הפיתוח (פיתוח חלונאי, פיתוח לאינטרנט, פיתוח שירותי רשת, פיתוח לטלפונים חכמים וטאבלטים, פיתוח לקונסולות משחק, משקפיי מציאות רבודה ....)

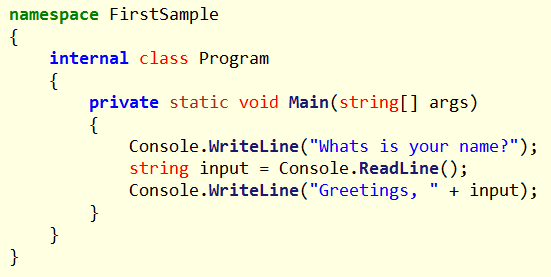
כמו בשפות האחרות שהוזכרו, מתכנתים בשפת C# אינם צריכים לכתוב ספריות עזר המספקות שירותים אלו אלא להתמקד אך ורק בלוגיקה של התוכנה מה שמאוד מקל ומאוד מזרז את פיתוח התוכנה, רק שבשפת C# הספריות הן חלק אינטגרלי בתשתית .NET Framework.

C++ היא שפת קומפיילר, Pyton היא שפת אינטרפרטר, C# היא משהו באמצע.  
הקוד שנכתב בשפת C# עובר קומפילציה אבל לא לשפת מכונה אלא לשפת ביניים הנקראת CIL (Common Intermediate Language), ה- CIL עובר סוג של הידור חכם לשפת מכונה בזמן ריצה על ידי רכיב מיוחד וחשוב הנקרא CLR (Common Language Runtime).

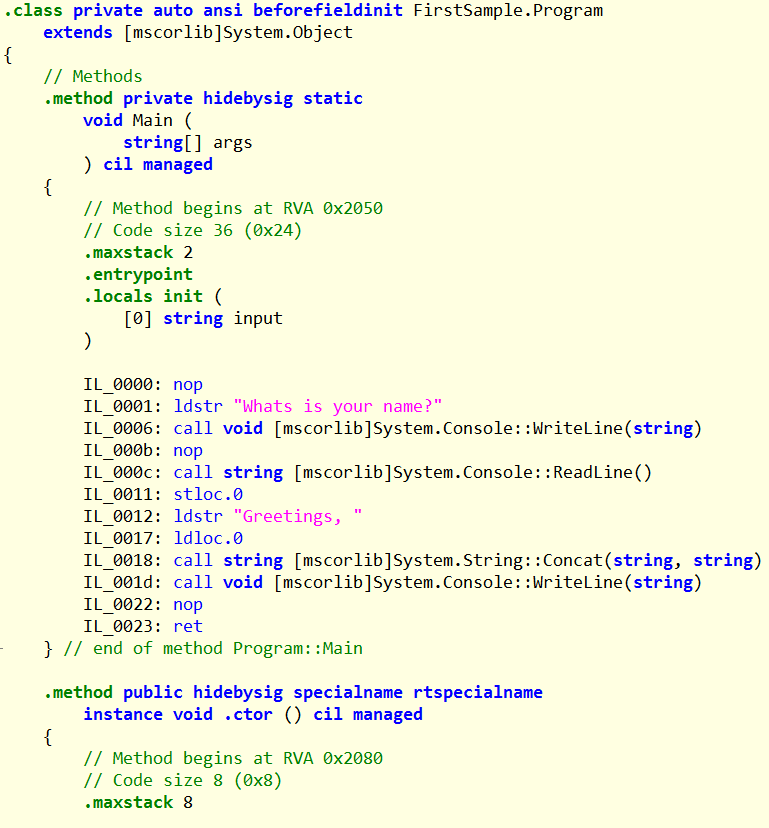


ה- CIL הוא קוד מקומפל אבל לא הוא לא שפת מכונה, לדוגמה:

הקוד הבא נכתב ב- C# (לא משנה מה הוא עושה):



לאחר הקומפילציה שפת הביניים (CIL) נראית כך (רק חלק ממנו כי הוא ארוך בהרבה):



ברור שזה לא בינארי וברור שהמהדר אינו יכול להריץ את זה בצורה הזו, נכון?

למה זה טוב? למה לא לקמפל ישר לבינארי?

יש לזה הרבה יתרונות שהיריעה קצרה בשביל להרחיב בהן, אולם אחת מהן היא Cross Platform, את ה- CIL ניתן להדר לשפת מכונה על כל סוג של מערכת הפעלה, כל מה שצריך הוא CLR ייעודי לאותה מכונה.

**דוגמה ראשונה**

שימוש במרחב השמות System

הגדרת מרחב שמות לפרויקט

המחלקה הראשית

|  |
| --- |
| using System;  namespace FirstSample  {  המתודה הראשית  class Program  {  static void Main(string[] args)  {  Console.WriteLine("Whats is your name?");  string input = Console.ReadLine();  Console.WriteLine("Greetings, " + input);  הקוד  }  }  } |

1. **מרחב השמות System** - כל טיפוס המוגדר ב- .NET Framework מוגדר בתוך רכיב לוגי הנקרא מרחב שמות (Namespace), במרחב שמות אחד נרכז אוסף של מחלקות, אשר קיים קשר לוגי ביניהן.  
   למה זה טוב? מאפשר להבחין ולהבדיל בין מחלקות שונות בעלות אותו השם, בעידן שבו אנו נשתמש בתשתיות שמישהו אחר כתב (תשתיות Open Source לדוגמה) יתכן מצב שתהיה התנגשות בין שמות מחלקות, מרחבי השמות פותרים בעיה זו.

מרחב השמות המרכזי והחשוב ב- .NET Framework הוא **System** , תחת מרחב שמות זה מוגדרות המחלקות (Classes) והמבנים (Structures) החשובים ביותר שנשתמש בהם בכל תוכנית מחשב שנכתוב ב- C#,   
על מנת שנוכל לנצל את המחלקות ושאר הטיפוסים המוגדרים במרחב שמות זה נצטרך להגדיר אותו , ההגדרה מתבצעת באמצעות המילה השמורה **using** .

1. **הגדרת מרחב שמות לפרויקט** - מקובל שגם את הקוד שאנחנו נכתוב נשייך למרחב שמות ששמו כברירת מחדל הוא כשם הפרויקט.
2. **המחלקה הראשית** – בכל תוכנית מחשב תהיה מחלקה המוגדרת כמחלקה הראשית, זו תהיה המחלקה בה תוגדר **המתודה הראשית** Main.
3. **המתודה הראשית Main היא static** (שימו לב, בניגוד ל-c היא נכתבת עם M גדולה)
4. **Console** – מחלקה חשובה המגדירה מספר רב של מתודות סטטיות המאפשרות לבצע קלט/פלט בחלון הקונסול.

**Pure Object Oriented**

C# היא שפה מונחת אובייקטים טהורה, המשמעות היא שכל מה הטיפוסים שמוגדרים בה הם אובייקטים,   
אפילו המשתנים הפשוטים הם אובייקטים ולא משתנים פרימטיביים כמו ב- C\C++, יש להם מתודות, לדוגמה:

|  |
| --- |
| int num = 123;  string str1 = num.ToString();  string str2 = 123.ToString();  num = int.Parse(str2); |

על מנת ליישם את העיקרון של Pure Object Oriented הטיפוסים הפשוטים אינם מוגדרים בשפה אלא הם חלק מה- .NET Framework.

**Garbage Collection**

יכולת שבה סביבת הריצה של תוכנית מחשב משחררת באופן אוטומטי זיכרון שהוקצה דינאמית ואין בו עוד צורך.

ב- C# נקצה זיכרון אולם, בניגוד ל-C++, אין צורך שנהיה עסוקים בשחרורו, מנגנון הנקרא GC יעשה זאת עבורנו.

לדוגמה:

|  |
| --- |
| Person p = new Person("031232344", "Moshe","Cohen",32);  p.AddFriend("034454444");  p.AddFriend("314468934");  p.Print(); |

ביצענו הקצאה לאובייקט מהמחלקה Person ובסיום העבודה עליו לא שחררנו אותו,   
אין צורך משום שה- GC יעשה את עבודת שחרור משאבי הזיכרון.

**מבוא ל-C#**

בקישור הבא מתפרסם [מדריך ללימוד השפה](http://www.completecsharptutorial.com/basic/getting-started/), עיברו במהירות (בדגש על התחביר) על חמשת הפרקים הראשונים:

* Start
* Variables and DataTypes
* Operators
* Conditional Constructs
* C# Statements
* Loop Constructs

**תזכרו** – אתם לא לומדים תכנות מחדש, את המלאכה הזו אתם כבר מכירים, אתם רק צריכים להכיר את התחביר הייחודי של השפה.

**הערה** – חמשת הפרקים הראשונים הם הבסיס לפיתוח בשפת C# , סביר מאוד להניח שבהמלך הפיתוח של ממשק המשתמש ב- WPF תצטרכו לחזור למדריך הזה עוד מספר פעמים על מנת להכיר נושאים נוספים שידרשו.