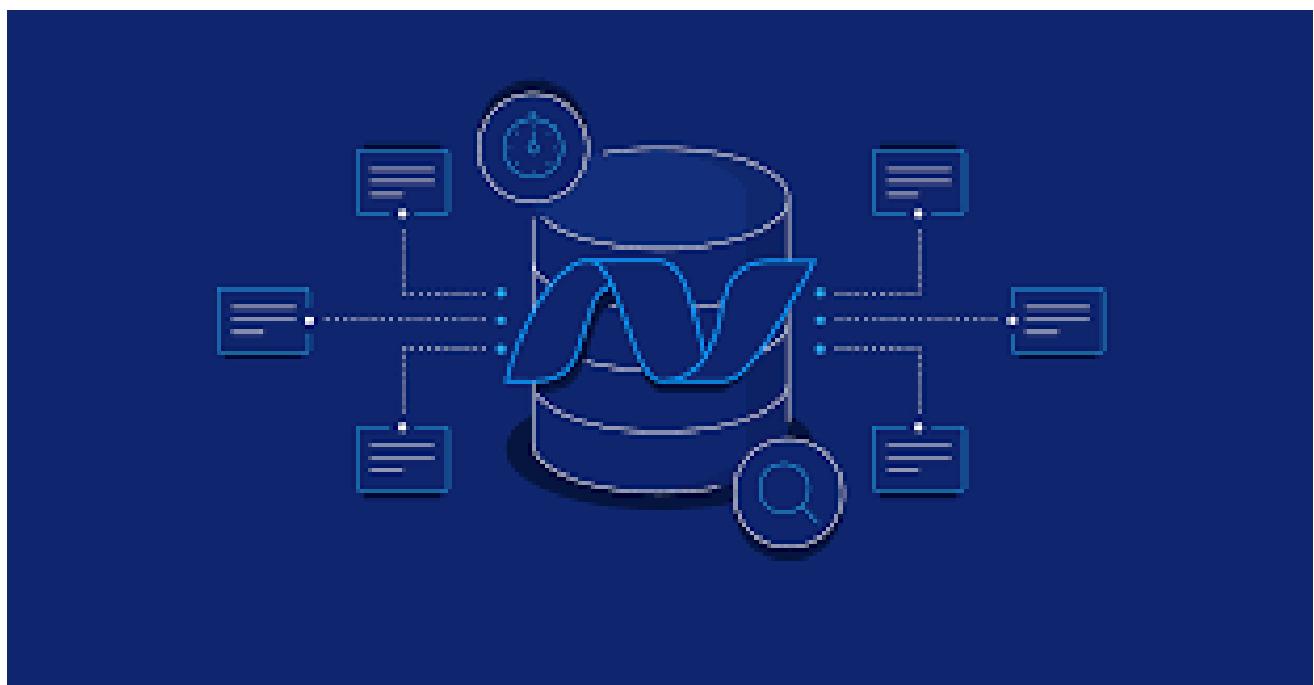


Advance Learn:

| | | |
|----------------|---------------------------|--|
| 3 - 45 | ASP.net + MVC + NG | תאורטי כה פיתוח פרויקט ASP יצירת - < MVC שאילות { API } הקי שור קומפוננטות הפעולות וההצגה |
| 46 - 98 | TypeScript | יש כן ?! הגדרת משתנים מערכות פונקציות = { [מערכים] פונקציות ^ & ^ כתיבה ^ & פונקציות \$ מיוחדות \$ סוגי הפניות מיוחדים בנייה מחלקה Abstract & Generic שדרוגי שן |





תיקיות בשימוש 2 ASP.NET ואילך [עריכה קוד מקור | עריכה]

- App_Code - עברו מחלוקת ומשתנים הנמצאים בשימוש בכל האתר ולא בדף בודד
 - App_Data - עברו נתונים
 - App_Browsers - עברו התאמת אישית לדפדפניים שונים
 - App_WebService - עברו שימוש ב-WebServices
 - App_WebReferences - עברו שימוש בקובצי מושגים
 - App_GlobalResources - עברו שימוש בקובץ קובץ עיצוב
 - Theme - עברו שמייה מספר אופציות של תבניות עיצוב
 - Bin - תיקייה עברו קובץ.dll המכילים אובייקטים ואוטומציות מהודרות של asp.net.asp
- ספריות אלו לרוב מוגנות מהורדה על ידי משתמש קצר, ללא תלות בסוג הקובץ.
- קובציים אחרים נמצאים בשימוש סבירות הפיתוח ומוגנים ברמת השרת על מנת למנוע הורדה שלהם על ידי לקוחות (למשל vbproj, csproj).

תיקיות בשימוש ASP.NET MVC [עריכה קוד מקור | עריכה]

- Models - מצין את התקינה שבה נשמרים כל מבני הנתונים של האתר.
- Controllers - תיקייה קבועה שבה נמצאים כל controllers של האתר, שליטה בתגובה השירות.
- Views - תיקייה שלוחב כוללת תיקיות משנה עם שם הבקר (Controller) שהיא מאוחסנת כל דפי cshtml הרלוונטיים לו, ובנוסף תיקייה בשם Shared שבה מאוחסנים דפי cshtml המשמשים באופן גלובלי.

ASP.NET AJAX [עריכה קוד מקור | עריכה]

בתאריך 23 בינואר 2007, מיקרוסופט הוציא לאור גרסה 1.0 של **ASP.NET AJAX**, אשר מהווה חבילת הרחבה עם פונקציונליות Ajax.

אך 56 שפות ▾

AJAX (תכנות)

AJAX (ראשי תיבות של **Asynchronous JavaScript And XML**) היא טכניקה לייצור יישומי רשת אינטראקטיביים המבוססים על קוד המורץ במסגרת דף HTML בלבד, ולא כישום מרובה דפים, נמוך בסביבה ה-Web. מטרתה העיקרית של הטכניקה היא שיפור חוויית המשתמש והצתת מהירות הטעינה של דפי האינטרנט, לאחר שהייתה אפשרה עדין רק חלקים מוקשים בדף האינטרנט, ללא צורך לטען את הדף כולו מחדש במחשבו של המשתמש.

מטרה זו מושגת באמצעות יצירת תקשורת וחילוף מידע בין מחשב הלקוח לשרת דפי האינטרנט באמצעות קוד JavaScript. למשל, האינטראקטיביות של "ישומי AJAX" מושגת באמצעות קוד המורץ בצד-הלקוח, זאת בגין השימוש בטכנולוגיות כמו PHP ו-ASP, שבו הקוד מורץ בצד-השרת, ולוקה מגע דף HTML סטטי, ולא ישום מלא. באופן זה מנענת תקשורת מיותרת בין הלקוח לשרת, מופחת העומס בעד השירות ונפח המידע החזר על עצמו אשר נשלח ללקוח פוחת.

כאשר משתמשים ב-AJAX עדין יש צורך בטכנולוגיית צד-שרת כמו PHP או ASP, אלא שבמקום לשלוח דף HTML שלם, נשלח ללקוח רק המידע, והלקוח מפרש את המידע ומציג אותו למשתמש.

יתרונות וחסרונות [עריכה קוד מקור | עריכה]

יתרונות השיטה:

- מילוי טפסים ארכויים: כאשר משתמש מתחבק למלא טופס והשדות קשורים אחד לשני בקשרי גומלין, ניתן למשוך קשרי גומלין אלה בגוף הטופס. דוגמה: אם נסמן שם מדינה תיפרס לפניה לבחירתנו רשותה הערים שבה.
- עדכון ריצף של המידע המוצג, ללא צורך לרענן דף אינטרנט בכל פעם. כך למשל כאשר מנהלים שיחה בפורום ניתן לראות תשובות מיד כשהן מועלות על האתר.
- סקורי דעת קהל ודירוגים: התוכנה מ起來 את זמן הפעולה, כך שמצוצם זמן המתנת הגולשים והיענותם גדול.
- התרעה על טעויות: מאפשר לשמשים לגłówות מיידית טעויות ולא להקליד נתונים חדש (למשל: אם שם משתמש כבר קיים במערכת, תתקבל התרעה מיידית).
- שלמה אוטומטית של טקסטים: בעת הקלדת הטקסט המערכת מזהה מילים מוכחות ומשלימה אותן אוטומטית. לדוגמה: למנוע החיפוש של גוגל.

חסרונות השיטה:

- תפקוד לחצן Back של הדף יכול להיפגע
- קשיי בשימוש דפים ב-**毋עדפים**
- **השהייה הרשת** (network delay) יכולה לגרום לביעית שימושיות JavaScript
- יש צורך לאפשר **AJAX** (גאגן WAI)
- קשיי **בתוכננו נגישות** (גאגן WAI)
- עומס על בסיס הנתונים (במוקום שאלתה אחת, תבוצע שאלתה על כל גישת AJAX)

(הופנה מהדף .NET)

אין לבלבל ערך זה עם הערך ["NET Framework."](#)



.NET Core, בעבר נקראה (.NET Core), היא פלטפורמה חופשית מבית חברת התוכנה [マイクロסופט](#), מבוססת קוד פתוח וחוצה-פלטפורמות, המאפשרת פיתוח והריצת תוכנה. הפלטפורמה יודעה להחליף את [.NET Framework](#). והוא תומכת ב망אות רחבות של תוכנות שפותחו בסביבת .NET Framework. בתנאי שקווד המקור עבר הידור לדוחט נט.

שחרור Core.NET. היזה מהלך מהפכני באסטרטגייה הרכזות של "מיקרוסופט" מפני שהוא מאפשר הרצת קוד שפותה עבור.NET. לא רק על "חלונות" אלא גם על Linux-MacOS, מכשירים חדשים וכיוצא באלו. על ידי כך, למרות החיסרונו של פגעה אפשרית ברכיש של מערכת הפעלה "חלונות", למעשה מפיצה על כך התייחסן של הרחבת שוק היעד באופן משמעותי.

של תוכנות שפותחו בארכיטקטורת.NET. והו נחלתם הבלתיודית של משתמשי "חלונות".

| | |
|-----|------------------|
| 1 | ארכיטקטורה |
| 1.1 | רכיבים מרכזיים |
| 2 | גרסאות |
| 3 | ראו גם |
| 4 | קישורים חיצוניים |
| 5 | הערות שוליים |

עריכה | מקור קוד קוד הערך

הגם ש-.NET Core. חולקת ת-קבוצה של ממשקי ה-API עם ה-.NET Framework, היא מגיעה עם הרחבות משלها שאין כלולות ב-.NET Framework., יתרה מכך, היא מכילה את CoreRT לתוכה בהידור בתכורת "Ahead-of-time" של .NET.

.NET Core. תומכת באربع פלטפורמות או סביבות:

- ASP.NET Core - פיתוח אפליקציות מקוונות
 - אפליקציות מוגשות **שורת פקודה** (Command Line)
 - ספריות (Class Libraries) או **ibraries**
 - Universal Windows Platform - אפליקציות מוגשות **UWP**
 - Windows Forms ו-WPF - ניהול מסגרת 3.0 [1]

רכיבים מרכזיים [עריכת קוד מקור | עריכה]

- CoreCLR - בדומה ל-CLR של .NET Framework, מכונה וירטואלי שמרתח תוכנות.NET. ותומכת באופן מלא ב-.CLR.
 - RyuJIT - מודול היפר-יעילות החדש שלCLR.NET, שיפורו של JIT מודול היפר-יעילות שלCLR.NET.
 - CoreFX - פיטול מצלג חלקו המרכזי של.NET Framework.

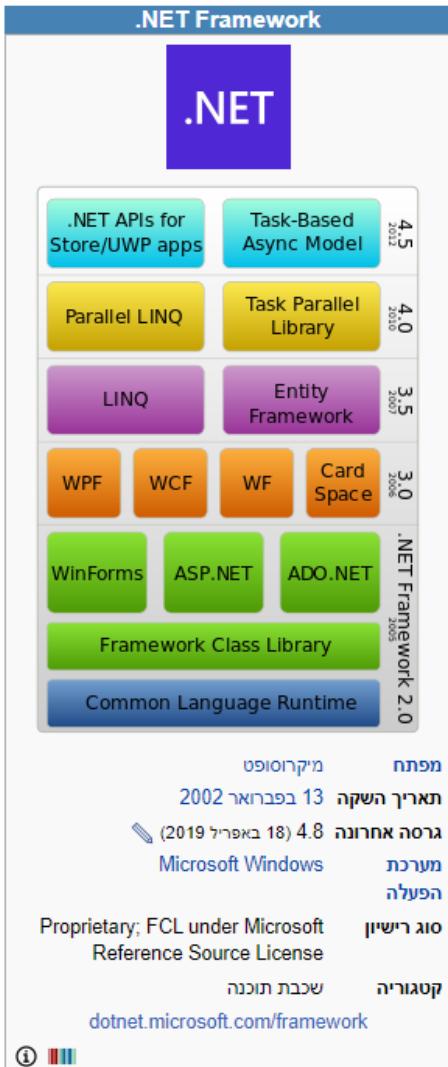
גרסאות [עריכה קוד מקור | עריכה]

| מספר הגרסה | תאריך שחרור | חידושים ועדכונים |
|---------------|--------------------|---|
| NET Core .1.0 | 2016-06-27 | הגרסה הרשמית הראשונה של Core .NET. |
| NET Core .1.1 | 2016-11-16 | תאימות ל-Mac OS Sierra (שוחררה יחד עם Visual Studio 2017). |
| NET Core .2.0 | 2017-08-14 | תמיכה ב-Razor Pages, הידור ייחד למספר רב של מערכות LINQ-to-SQL, ותמיכה מובנית בкриיפטוגרפיה. ^[5] |
| NET Core .2.1 | 2018-05-30 | שחרור אפליקציות עם זמן ריצה מובנה, וכן דחיסה גבוהה של קבצים. ^[6] |
| NET Core .2.2 | 2018-12-04 | האזנה לאירועים בזמן הריצה. בנוסף ישנה תמיכה בהידור תליי מסגרת - שיהיה קטן יותר אך ישמש בשירותי זמן הריצה שימושיים על המערכת. |
| NET Core .3.0 | 2019-09-23 | תמיכה באפליקציות שלוחן עבודה ל-"חלונות" לראשונה, אפליקציה בת קוד ייחוד, תמיכת C# 8.0+, שיפור מהירות ה-Host, תמיכה ב-HTTP/2, עדכון ושיפור ספריות הкриיפטוגרפיה, אפשר גישה ליציאה טורית, ותמיכה במערכות מבוססות ARM64. ^[7] |
| .NET 5.0 | 2020-11-10 | החלפה של NET Framework. באופס סופי (אך עדין ישנה תמיכה בה) - על כן ישנה "קפיצה" בגרסאות (שהרי הגרסה האخيرة של NET Framework היא 4.8), ושיפור ממשוני במערכות של ספריית ה-JSON. ^[8] |
| .NET 6 | 2021-11-08 [10] | 17.0 Visual Studio 2022 גרסה |

.NET Framework

(.NET Framework) הופנה מחדף ערך זה עם המילה ".NET".

אין בבלבול ערך זה עם המילה ".NET".



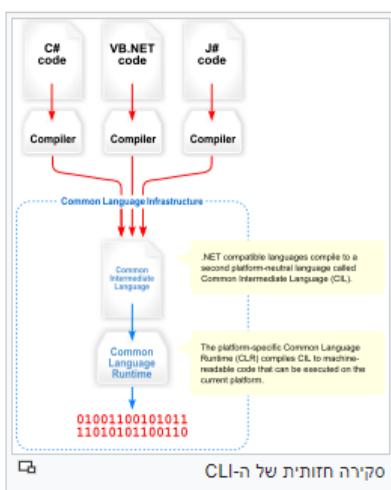
.NET Framework. ("דוט נט פרימורק" או "תשתיות דוט נט") היא שכבת תוכנה של Microsoft שבאמת עצמה מקודמת ומורכצת תוכנות אחרות. שכבה זו מספקת לתוכנות שרכזות מעלה סט כלים ושירותים כגון: ניהול זיכרון, שילטה בהתקני GPU/פלט, הפעלת הודעות וכו'. כך שהתוכנות אינם צריכים לכתוב ספריות עדר המספקות שירותים אלה, אלא רק את הלוגיקה העיקרית של התוכנה. בנוסף, שימוש בשכבת תוכנה זו (הנקראת גם "סביבה וירטואלית") מאפשר להריץ תוכנות על פלטפורמות שונות: חלונות, לינוקס, יוניקס, טאבלטים, טלפונים סולריים וכו', בלי צורך בשינוי קוד התוכנה, כי התוכנה "מדברת" רק עם הסביבה הוירטואלית, וביביה זו כבר יודעת איך לדבר עם הפלטפורמה הספציפית. **.NET.** הושקה ב-11 בפברואר 2002^[1].

הפלטפורמה מספקת ממשק פיתוח אחדן לתוכנות שלוחניות, פיתוח אינטרנט Web, והן לתוכנות לסמארטפונים המרצים Windows Phone ואך אפשרית לקשרו ביחס המתכוון מחשב שוני ולשפת ביניהם ישומים ומייד. היא פותחה כמענה תחרותי לארכיטקטורת J2EE מבית סאן מיקרוסיסטם. בתחום התוכנות השולחניות, מספקת.NET. ספרייה אחת לפלטפורמת ממשק הפעלה חלונית, החל מגרסת 98 Windows, במקומם משמשי תכונות היישומים של MFC.

עד שנת 2016, יצאו עשרה גרסאות של פלטפורמה זו. בראש צוות הפיתוח עמד היישראלי יובל נאמן, לשעבר סגן נשיא חטיבת כל הפיתוח במיקרוסופט העולמית. שמה של הטכנולוגיה נלקח מסימנת האינטנס שמצוינת אונרדים רשותים, וזאת מכיוון שהטכנולוגיה פותחה בשיא בעת האינטרנט (אם כי גרסתה הרשמית הראשונה, גרסה 1.0, הוכרזה לאחר התפוצצות הבועה, בתחילת 2002). פלטפורמת פיתוח זו תומכת במגוון רב של שפות תכנות, ביניהם: C#, VB.NET, F#, שפת Boo ו-Vala. וגם מחללי ישומים כדוגמת PowerBuilder וסביבות כמו Delphi, מאפשרים עבודה בסביבה זו.

תיקון עניינים [הסתנה]

- יכולות ומרכזים עיקריים
- פרשה ושיכון
- גרסאות
- יתרונות
- חרונות
- הדרים
- ניהול זיכרון
- ראו גם
- קישורים חיצוניים
- הוסף על ידי: מילא כהן



• **תפוקוליות בינה (Interoperability):** לאחר שלעיטים קרובות בדרשת יכולת לתקשר עם תוכניות ישנות שלא פותחו ב-

-NET. סביבת ה-.NET. מאפשרת להשתמש בפונקציונויות שבובעת בתוכניות שפותחו מחוץ לסביבת.NET. גישה לרכיב COM מהאפשרה באמצעות מרחב ה-System.Runtime.InteropServices. גישה לרכיב COM מהאפשרה באמצעות System.EnterpriseServices וניתן לגשת לפונקציות אחרות באמצעות System.EnterpriseServices.

• **מנוע (JIT):** שפה התוכנות בסביבת.NET. מהודרת לשפת ביניים הנקראת Common Intermediate Language (CIL). בימוש של Microsoft שפת ביניים זו אינה מפורשת אלא מהודרת בזמן ריצה (just-in-time; JIT), שאוטו מימושה Microsoft-.NET. לשפת מכונה. צירוף רעיונות אלו נקרא Common Language Infrastructure (CLI).

• **אי תלות בשפה:** סביבת.NET. משתמשת בתוך הנקרא Common Type System ובקיים CTS. תוך זה מגדיר את כל טיפוסי הנתונים והמושגים התכונתיים הנתמכים על ידי CLR וכיום הם יתקשרו או לא יתקשרו זה עם זה. הודות לכך מאפשרת סביבת.NET. המרת מופעים של טיפוסים בין שפות שונות שנתמכות ב-NET.

• **ספריית מחלקות בסיסית (Base Class Library):** ספריית מחלקות בסיסית הנקראת Base Class Library. ב-BCL היא חלק מ-.NET Framework Class Library. היא ספרייה של מחלקות המספקת פונקציונליות שניתן להשתמש בהן בכל שפה.NET. ספרייה זו עוצפת פונקציות נפוצות כדוגמת קרייה וכ כתיבה של קבצים, ציר גרפי, תקשורת עם מסד נתונים ועובדת עם XML.

• **niedות:** התכוון של סביבת.NET. מאפשר ברמת התאוריה לפתוח תוכנה שלא תהיה בפלטפורמה ושתעבד

במגוון פלטפורמות ללא צורך בהתחמת הקוד לכל אחת ואחת. Microsoft פיתחה תמייה בסביבת.NET. בעבר Windows, Windows CE, Xbox 360, ו-Xbox. הפירוט של -.NET. וספרות התוכנות שלה שיתאים לפלטפורמות אחרות.

יתרונות [עריכת קוד מקור | עריכה]

- תמייה בפרוטוקול **SOAP** של ה-W3C, שמאפשר הפעלה ללא קשר לשפה מעבר ל**HTTP**.
- תמייה בחלוקת משימות בין שרת לתוכנת לקוח כגון דףדף.
- תמייה בקריאה וכתיבה של קובץ **XML**.
- ממשק משתמש גרפי חלוני. בשימוש של מיקרוסופט, מדובר במקרה העומד לפונקציות ה-C של Windows עצמה האחראיות לייצירת ממשק המשתמש. בשימוש של "mono", ממשק המשתמש מziejיר על ידי אובייקט צייר הגרפי של **.NET**.
- קריאה-ל-**API** של מערכת הפעלה מתוך התוכנית בצורה כמעט טبيعית. פועלה לא טריויאלית, בהתחשב בכך שהיא API כתוב בשפת C, בעוד שפות **.NET** הן שפות אחרות, מוכוונות עצמאיות.
- אובייקטים מסווגים שהן משתמשות כל השפות התומכות בסביבה זו.

חרוגנות [עריכת קוד מקור | עריכה]

- אחד החסרונות העיקריים של הסביבה הוא שקוד ההרצה שהוא קוד בינוני קרי, שאפשר בכלים פשוטים להמירו לכל שפה ב-.NET., כך שהגינה על אלגוריתם והאכיות של מפתחיו, כמו גם חשיבות עיקפת מגנון בקרת רישיון תוכנה, הפכה להיות קשה יותר, מאשר בקוד מהודר של שפה טبيعית. שיטת ההגנה המרכזית נקבעת עירוף (obfuscation) (או), שבו תוקן הקוד משונה באמצעות ומספרים חסרי משמעות. חברת מיקרוסופט נתנה את דעתה על בעיה זו והזיאה עם חבילת הסטודיו גרסה חדשה של התוכנה בשם **Dotfuscator**, שיעודה לערפל קובץ הריצה (**EXE**, **DLL**) שהודор בסביבת ה-.NET.
- יש הטוענים נגד הנפק הרבה של גרסאות הרצה בגרסאות 3 של ה-.NET. בחלק ממערכות הפעלה כדוגמת חלונות XP, כדי להריץ במחשב לקוח אפלו יישום קיטן של ה-.NET. בגודל של 100K שמשתמש בתקשורת WCF, יש צורך להוריד ולהתקין קובץ הפעלה בנפח של מאות מגהבייט. (מכיון שככל גרסה גבוהה אחת על השנייה, מי שרצה להשתמש בתוכנות המעודכנות בגרסה 3.5. צריך להתקין גרסה 3 בנפח 50 מגהבייט, גרסה 3.5 מגהבייט ועוד 250 מגהבייט-ל-.NET 3.5 SP1). בגרסה 4 של ה-.NET. בעה זו תוקנה בחלוקת (ונפח קובץ הפעלה שאינו תלוי בגרסאות קודמות, עומד על 48 מגהבייט-ל-x64+x86). פתרון נוסף לבעה זו הוצע בגרסה 4 של ה-.NET. והציג אפשרות לריצה תחת תתקבוצה של ריבוי מערכות המונינה Client Profile. ריצה בפרופיל זה מוגבלת לשימוש בספריות התשתיות הנפוצות באפליקציות לקוח ובכך מאפשרת הקטנה של נפח קובץ ספציפיות הריצה שהמשתמש מחיב להתקין.
- בעוד שטבעם של התקנים שמרכבים את ה-.NET. להיות חוץ פלטפורמות, ישות מלא של Windows. מיקרוסופט מספקת תמייה מוגבלת עבור פלטפורמות אחרות כגון XNA עבור Xbox 360 ו-Xbox Phone 7, Silverlight ו-Mac OS, Windows Store ועוד. קיימים מימושים חלופיים של CLR, ספריות מחלקה בסיס ומדרים (לפעמים של ספקים אחרים), אך בזמן שככל המימושים הללו מבוססים על אותו תקנים, הם עדין מימושים שונים עם רמות שונות של שלמות, בהשוואה לגרסה המלאה של ה-.NET. המשמעות לכך מיקרוסופט.

מהדרים [עריכת קוד מקור | עריכה]

- ב-2 באפריל 2014 מיקרוסופט הוציאו לאור גרסת בטא למהדר חדש בשם **C#-Native**^[3], המאפשר להדר קוד חדש בשפת C# ו-BB.NET DotNET Native (מהדר דינמי – Just-In-Time Compiler). לפיו מיקרוסופט, מהדר זה מתבסס על המהדר של ויזואל סטודיו לשפת C++, והוא מהדר סטטי. מיועד לישומי Windows Store בלבד (ישומים למחשבים לולוי מבוססים על אותו תקנים, הם עדין מימושים שונים עם רמות שונות של שלמות, בהשוואה לגרסה המלאה של Applications). אך הדבר לא התרחש עד הרבעון הראשון של 2016.
- היתרון הבולט ומהדר זה הוא הביצועים – ניתן לכתוב קוד בשפת C# עם ביצועים דומים לקוד בשפת C++, לפיו מיקרוסופט. בנוסף לכך, מהדר זה מאפשר פלטפורמות שונות, ומסוגל לעוננו לשפת המקור של הקובץ מהודר (Assembly Executable). יתרון נוסף הוא שגם על פי שהקוד הוא קוד מקורה, ניהול הדיכון האוטומטי ושאר היתרונות של .NET. עדין מתאפשרים באופן מלא לחלוטין, בינו לבין שפות אחרות שמתהדרות לקוד מקורה (כדוגמת C++). מהדר זה לא יחליף את מהדר JIT, אך הוא יספק חלופה למהדר JIT (ניתן לקבוע את סוג ההידור, JIT או Native, לפני ההידור).

ניהול זיכרון [עריכת קוד מקור | עריכה]

ניהול הזיכרון של שפות המפותחות בסביבה זו נשען באופן אוטומטי ביכולת לתכנון בשפה כדוגמת C++.CLR מספק שירותים לטיפול בנושא של הקצת זיכרון דינמית. אובייקטים שהוקצו אינם דורשים נקי ידי בתום השימוש ויש מערכת של "איסוף זבל" שודאגת לפונת זיכרון של אובייקטים שכבר אינם בשימוש.

שניהם המושנים שתפנסו לא מעת בלימודי התכונות הם פרימורק (Framework) וספריה (Library). בעיקרן מרבית הספריות והפרימורקים הם קודים לשימוש חוזר. הקודים האלה אמנים נכתבו על ידי אחרים, אבל יכולים לשיער לכם לפחות בבעיות נפוצות בקוד שלכם.

אבל מהם השניהם ומה ההבדל ביניהם?

הבדלים

ספריות ופרימורקס (בעברית מסגרות) הן שתיהן רכיבי קוד שנכתב על ידי אחרים ומשמש לפתרון בעיות נפוצות.

פתחים משתמשים בדרך כלל לטיורון במונחים "ספריה" (Library) ו"פרימורק" (Framework), אבל יש ביןיהם הבדל מהותי שכדי להבין.

אם אתם מתכוונים ויצא לכם ליצור פונקציה, כדי שתתבצע עבורם העבודה, או ברכותינו - נס אם היא שלכם ונכתבה על ידכם, יצרתם ספריה!

פרימורק היא מסגרת תכניתית שאתם בוחרים והוא תקבע לכם את האופן בו תעבדו ותפתחו, בתמורה לקלות וליד מכונת שהיא תציע לכם.

המחשת הבדלים

בואו נדמה את השניהם לדברים שאתם מכירים מהבית שלכם:

ספריה היא כמו ל在京ת רהיטים מ-Ikea. נניח שיש לכם בית (במטפורה זהו הקוד שלכם) ואתם צריכים עזרה ברהיטים שיהיו בו. במקום שתצטרכו לבנות שולחן מאפס, כלומר לכתוב הכל בעצמכם, איקאה מציעה לכם שלוחנות מוכנים שרק צריך להביא ולהרכיב בבית שלכם. עם הספריות אתם בשליטה ובקוד, בניגוד לאיקאה, זה איפלו בחינם!

פרימורק, לעומת הספריה, היא יותר כמו לבנות מודל של בית. ניתן לדמות את הפרימורק לסדרה שלشروط ולבחר אפשרויות מוגדר של האדריכלות והעיצוב שלו. עם פרימורק לא תוכל לעשות כל מה שברצונכם, אבל זה גם יתרון כי השליטה היא בידי והוא יבקש מכם לתת את מה שהוא צריך ויבנה לכם את מה שאתם צריכים.

הבדל הטכני ביןיהם

הבדל הטכני בין פרימורק לספריה קשור ב"הופיע שליטה". אם בספריה השליטה היא בידי המתכנת שמשתמש בה כרצונו, בפרימורק השליטה היא בידי הפרימורק.

בשימוש בספריה, המתכנת הוא האחראי על זרימת היישום. הוא שבודор מתי ואיפה להתחבר לספריה. כמשמעותם בפרימורק, היא שஅחראית על הזרימה. כי הפרימורק מיע למתקנת כמה מקומות לחבר את הקוד אבל זה הוא שקורא לקוד הזה לפי הצורך.

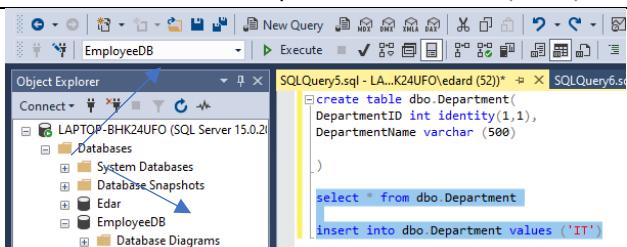
סיכום

פרימורק וספריות הם שניים קוד שנכתב על ידי אחרים ונitin להסתיע בו כדי לבצע משימות נפוצות עם פחות קוד.

הפרימורק לוקח את השליטה בתוכנית ואומר למתכנת מה הוא צריך. הוא דעתן ונוטן למפתח פחות חופש, אבל בתמורה מספק לו בטיחון וקלות בקידוד.

בספריה זה הפוך. המפתח מתקשר לספריה ומשתמש בה כשהוא זוקק לה. למקודם יש כאן הרבה יותר חופש, אבל גם יותר אחריות וקידוד.

פתיחת פרוייקט – :asp

| | |
|---|--|
| <pre>Create Database EmployeeDB create table dbo.Department(DepartmentID int identity(1,1), DepartmentName varchar (500)) insert into dbo.Department values ('IT')</pre> | יצירה DB יצירה Table |
|  | הכנסת ערכים בחירה שורות להרצת |

 ASP.NET Web Application (.NET Framework)

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

C# Windows Cloud Web

for creating applications that use the latest standards.

 **Web API**

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Advanced

Configure for HTTPS
 Docker support
(Requires Docker Desktop)
 Also create a project for unit tests

 **Single Page Application**

A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single

קונפיגורציה :

The screenshot shows the Visual Studio IDE with the code editor displaying the `Program.cs` file of a .NET Core Web Application. The code is annotated with arrows pointing to specific sections of the configuration logic.

```

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swash
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddCors(c =>
{
    c.AddPolicy("AllowOrigin",
        options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
});

var app = builder.Build();

app.UseCors(options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseAuthorization();

app.MapControllers();

app.Run();

```

Annotation 1: Points to the `builder.Services.AddCors(c =>` line. A callout box highlights the `builder.Services.AddCors(c =>` line and its contents.

```

builder.Services.AddCors(c =>
{
    c.AddPolicy("AllowOrigin",
        options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
});

```

Annotation 2: Points to the `options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();` part of the `AddCors` call. A callout box highlights this line.

```

options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();

```

Annotation 3: Points to the `app.UseCors(options =>` line. A callout box highlights this line.

```

app.UseCors(options =>
options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

```

Annotation 4: Points to the `services.AddCors(c =>` line in the `Configure` method. A callout box highlights this line.

```

services.AddCors(c =>
{
    c.AddPolicy("AllowOrigin",
        options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
});

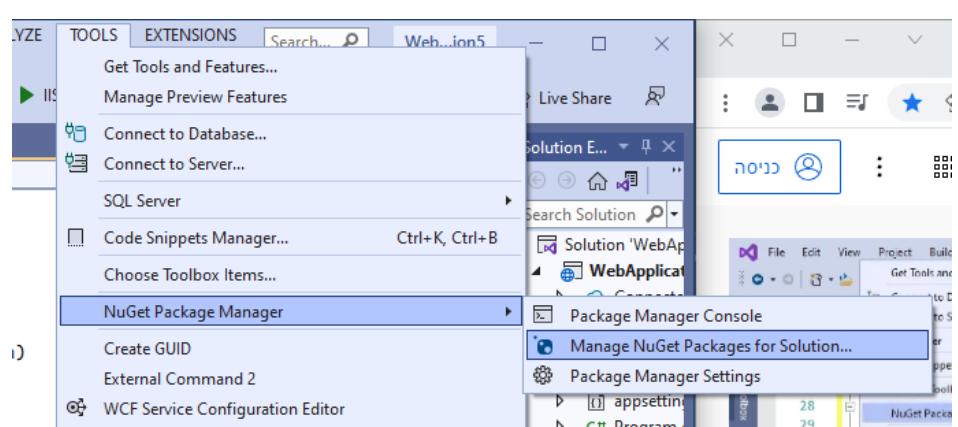
```

Annotation 5: Points to the `app.UseCors(options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());` line in the `Configure` method. A callout box highlights this line.

```

app.UseCors(options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

```



The screenshot shows the NuGet Package Manager interface. In the search bar at the top, the text 'mvc.newtonsoft' is entered. To the right of the search bar are three icons: a red 'X' for clearing the search, a blue circular arrow for refreshing, and a checkbox labeled 'Include prerelease'. Below the search bar, the results are displayed. A purple '.NET' badge is on the left. The first result is 'Microsoft.AspNetCore.Mvc.NewtonsoftJson 6.0.4', which is described as 'ASP.NET Core MVC features that use Newtonsoft.Json. Includes input and output formatters for JSON and JSO...'. A vertical scroll bar is visible on the right side of the results list.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swash
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddCors(c =>
{
    c.AddPolicy("AllowOrigin",
        options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
});

builder.Services.AddControllersWithViews().AddNewtonsoftJson(options =>
    options.SerializerSettings.ReferenceLoopHandling = Newtonsoft
        .Json.ReferenceLoopHandling.Ignore)
    .AddNewtonsoftJson(options => options.SerializerSettings.ContractResol
        = new DefaultContractResolver());
```

using Newtonsoft.Json.Serialization;

```
services.AddCors(c =>
{
    c.AddPolicy("AllowOrigin",
                options => options.AllowAnyOrigin().AllowAnyMethod().Allow
            });

services.AddControllersWithViews().AddNewtonsoftJson(options =>
options.SerializerSettings.ReferenceLoopHandling = Newtonsoft
.Json.ReferenceLoopHandling.Ignore)
.AddNewtonsoftJson(options => options.SerializerSettings.ContractResolver
= new DefaultContractResolver());

services.AddControllers();
```

services.Add

DB DW

"ConnectionStrings"

The screenshot shows the contents of the `appsettings.json` file. It includes the following sections:

```

{
  "ConnectionStrings": {
    "EmployeeAppCon": "Data Source =.; Initial Catalog=EmployeeDB; Integrated Security=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}

```

Two arrows point from the text "DB DW" and the text "ConnectionStrings" to the "ConnectionStrings" section of the JSON file.

: MVC צירת

The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer displays two model classes: `Department` and `Employee`, both located in the `WebApplication6.Models` namespace. The `Employee` class has several properties: `EmployeeId`, `EmployeeName`, `Department` (which is a foreign key), `DateOfJoining`, and `PhotoFileName`. A tooltip on the right indicates that the `Department` property is being used for database relationships.

On the right, the Object Browser shows the `Models` folder containing the `Department` class. Below it, the `Controller` context menu is open, with the "Add" option selected. A submenu shows options like "MVC Controller with read/write actions" and "API Controller". The "Controller..." option is highlighted.

A tooltip at the bottom right says "Controller בקשה Api" (Create Controller API).

```

namespace WebApplication6.Models
{
    public class Department
    {
        public int DepartmentId { get; set; }
        public string DepartmentName { get; set; }
    }

    public class Employee
    {
        public int EmployeeId { get; set; }
        public string EmployeeName { get; set; }

        public string Department { get; set; }

        public string DateOfJoining { get; set; }

        public string PhotoFileName { get; set; }
    }
}

```

Data Tipol
בפונקציות lbs



```
using System.Data.SqlClient;
using System.Data;
```

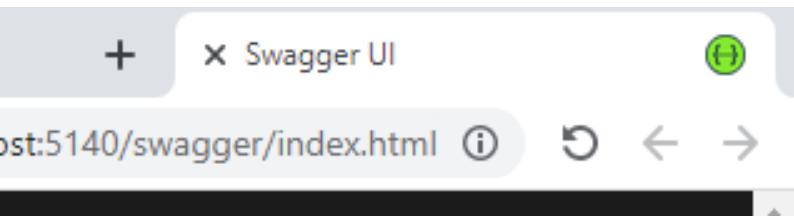
```
namespace WebApplication6.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class DepartmentController : ControllerBase
    {
        private readonly IConfiguration _configuration;

        0 references
        public DepartmentController (IConfiguration configuration)
        {
            _configuration = configuration;
        }
    }
```

privat

שאיילתות : Api

```
[HttpGet]
0 references
public JsonResult Get()
{
    string query = @"select DepartmentId,DepartmentName from dbo.Department";
    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using(SqlConnection myCon= new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query,myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
        }
    }
    return new JsonResult(table);
}
```



נתחבר

נבדוק בבקשת ב postman לфи
השם URL , שם קונט롤

http://localhost:5140/api/department

GET http://localhost:5140/api/department

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

| | KEY | VALUE | DESCRIPTION | ... | Bulk Edit |
|--|-----|-------|-------------|-----|-----------|
| | Key | Value | Description | | |

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "DepartmentId": 1,
4     "DepartmentName": "IT"
5   },
6   {
7     "DepartmentId": 3,
8     "DepartmentName": "Support"
9   }
10 ]

```

200 OK 230 ms 236 B Save Response

using WebApplication6.Models;

```
[HttpPost]
public JsonResult Post(Department dep)
{
    string query = @"insert into dbo.Department values ('"+dep.DepartmentName + @"')";
    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
        }
    }
    return new JsonResult("Added Successfully");
}
```

[HttpPost]

POST http://localhost:5140/api/department

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1
2"DepartmentName":"a"
3

Text
JavaScript
JSON
HTML

```
[HttpPost]
0 references
public JsonResult Put(Department dep)
{
    string query = @"update dbo.Department set
        DepartmentName = '" + dep.DepartmentName + "'"
        where DepartmentId = '" + dep.DepartmentId + "'";
    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
        }
    }
    return new JsonResult("Update Successfully");
}
```

[HttpPost]

PUT <http://localhost:5140/api/department> Send

| Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings | Cookies |
|--------|---------------|-----------------------|-------------|--------------------|---------|----------|----------|
| none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON | Beautify |

```

1 { "DepartmentId":1,
2   "DepartmentName": "WinGata"
3 }
```

Body Cookies Headers (4) Test Results 200 OK 256 ms 168 B Save Response

| | | | | | | |
|--------|-----|---------|-----------|------|------|--------|
| Pretty | Raw | Preview | Visualize | JSON | Copy | Search |
|--------|-----|---------|-----------|------|------|--------|

```
1 "Update Successfully"
```

```
[HttpDelete]
0 references
public JsonResult Delete(int id)
{
    string query = @"
        delete from dbo.Department
        where DepartmentId = " + id + @""";
    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
        }
    }
    return new JsonResult("Delete Successfully");
}
```

button1

http://localhost:5140/api/department

DELETE http://localhost:5140/api/department **Send**

Params Authorization Headers (8) Body **JSON** Cookies

none form-data x-www-form-urlencoded raw binary GraphQL Beautify

```
1
2 {"DepartmentId":5}
3
```

Body Cookies Headers (4) Test Results **200 OK** 217 ms 168 B Save Response

Pretty Raw Preview Visualize JSON

```
1 "Delete Successfully"
```

```

namespace WebApplication6.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class EmployeeController : ControllerBase
    {
        private readonly IConfiguration _configuration;

        0 references
        public EmployeeController(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        [HttpGet]
        0 references
        public JsonResult Get()
        {
            string query = @"
                select EmployeeId,EmployeeName,Department,
                       convert(varchar(10),DateOfJoining,120) as DateOfJoining,
                       PhotoFileName
                from dbo.Employee";
            DataTable table = new DataTable();
            string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
            SqlDataReader myReader;
            using (SqlConnection myCon = new SqlConnection(sqlDataSource))
            {
                myCon.Open();
                using (SqlCommand myCommand = new SqlCommand(query, myCon))
                {
                    myReader = myCommand.ExecuteReader();
                    table.Load(myReader);
                    myReader.Close();
                }
            }
            return new JsonResult(table);
        }
    }
}

```

[HttpGet]

```

{
    "EmployeeId": 1,
    "EmployeeName": "Sam",
    "Department": "IT",
    "DateOfJoining": "2020-06-01",
    "PhotoFileName": "anonymous.png"
}

```

```
[HttpPost]
0 references
public JsonResult Post(Employee emp)
{
    string query = @"insert into dbo.Employee
                      (EmployeeName,Department,DateOfJoining,PhotoFileName)
                      values
                      (
                          '" + emp.EmployeeName + @"'
                          ,'" + emp.Department + @"'
                          ,'" + emp.DateOfJoining + @"'
                          ,'" + emp.PhotoFileName + @"'
                      )";
    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
        }
    }
    return new JsonResult("Added Successfully");
}
```

```
[HttpPut]
0 references
public JsonResult Put(Employee emp)
{
    string query = @"
                    update dbo.Employee set
                    EmployeeName = '" + emp.EmployeeName + @"'
                    ,Department = '" + emp.Department + @"'
                    ,DateOfJoining ='" + emp.DateOfJoining + @"'
                    ,PhotoFileName ='" + emp.PhotoFileName + @"'
                    where EmployeeID = " + emp.EmployeeId + @"";
    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
        }
    }
    return new JsonResult("Update Successfully");
}
```

POST http://localhost:5140/api/employee

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```

1  {
2    "EmployeeName": "Tehila",
3    "Department": "MVC",
4    "DateOfJoining": "1998-12-13",
5    "PhotoFileName": "love"
6  }
7
8

```

http://localhost:5140/api/employee

PUT http://localhost:5140/api/employee

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```

1  {
2    "EmployeeName": "Galit1",
3    "Department": "Json",
4    "DateOfJoining": "1998-12-11",
5    "PhotoFileName": "aaa.mp3",
6    "EmployeeId": 222
7  }
8

```

Body Cookies Headers (4) Test Results 200 OK

Pretty Raw Preview Visualize **JSON**

```

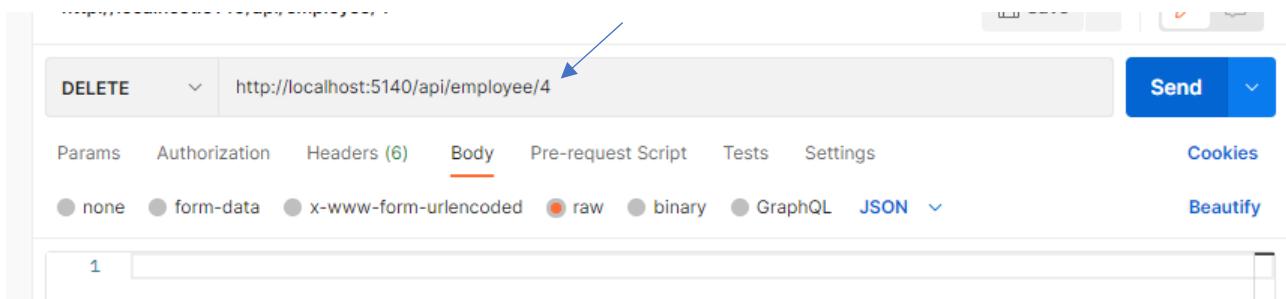
1  "Update Successfully"

```

```

[HttpDelete("{id}")]
public JsonResult Delete(int id)
{
    string query = @"
        delete from dbo.Employee
        where EmployeeID = " + id + @"";
    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
        }
    }
    return new JsonResult("Delete Successfully");
}

```



```
using System.IO;
using Microsoft.Extensions.FileProviders;
```

```
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(Directory.GetCurrentDirectory(), "Photos")),
    RequestPath = "/Photos"
});
```

app.Use

ניצור נתת תיקיה
Photos

```
using System.IO;
using Microsoft.AspNetCore.Hosting;
```

```
private readonly IConfiguration _configuration;
private readonly IWebHostEnvironment _webHostEnvironment;
```

pri

```
0 references
public EmployeeController(IConfiguration configuration, IWebHostEnvironment webHostEnvironment)
{
    _configuration = configuration;
    _webHostEnvironment = webHostEnvironment;
}
```

```
[Route("SaveFile")]
[HttpPost]
0 references
public JsonResult SaveFile()
{
    try
    {
        var httpRequest = Request.Form;
        var postedFile = httpRequest.Files[0];
        string filename = postedFile.FileName;
        var physicalPath = _webHostEnvironment.ContentRootPath + "/Photos/" + filename;

        using(var stream = new FileStream(physicalPath, FileMode.Create))
        {
            postedFile.CopyTo(stream);
        }

        return new JsonResult(filename);
    }
    catch(Exception)
    {
        return new JsonResult("anonymous.png");
    }
}
```

[Route(

POST http://localhost:5140/api/employee/SaveFile

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

| | KEY | VALUE | DESCR |
|-------------------------------------|------------|-------|-------------|
| <input checked="" type="checkbox"/> | myNewPhoto | Text | |
| | Key | Text | Description |
| | | File | |

POST http://localhost:5140/api/employee/SaveFile

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

| | KEY | VALUE | DESCRIPTION | ooo | Bulk Edit |
|-------------------------------------|------------|-------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | myNewPhoto | a.jpg | | | |
| | Key | Value | Description | | |
| | | | Description | | |

תינוקות נטיב

Route]

```
[Route("GetAllDepartmentNames")]
[HttpGet]
0 references
public JsonResult GetAllDepartmentNames()
{
    string query = @"
        select DepartmentName
        from dbo.Department";
    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
        }
    }
    return new JsonResult(table);
}
```

http://localhost:5140/api/employee/GetAllDepartmentNames

GET http://localhost:5140/api/employee/GetAllDepartmentNames Send

| Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings | Cookies |
|----------------------------|---------------------------------|---|--------------------------------------|------------------------------|-------------------------------|----------------------------|--------------------------------|
| <input type="radio"/> none | <input type="radio"/> form-data | <input type="radio"/> x-www-form-urlencoded | <input checked="" type="radio"/> raw | <input type="radio"/> binary | <input type="radio"/> GraphQL | <input type="radio"/> JSON | <input type="radio"/> Beautify |

1

Body Cookies Headers (4) Test Results

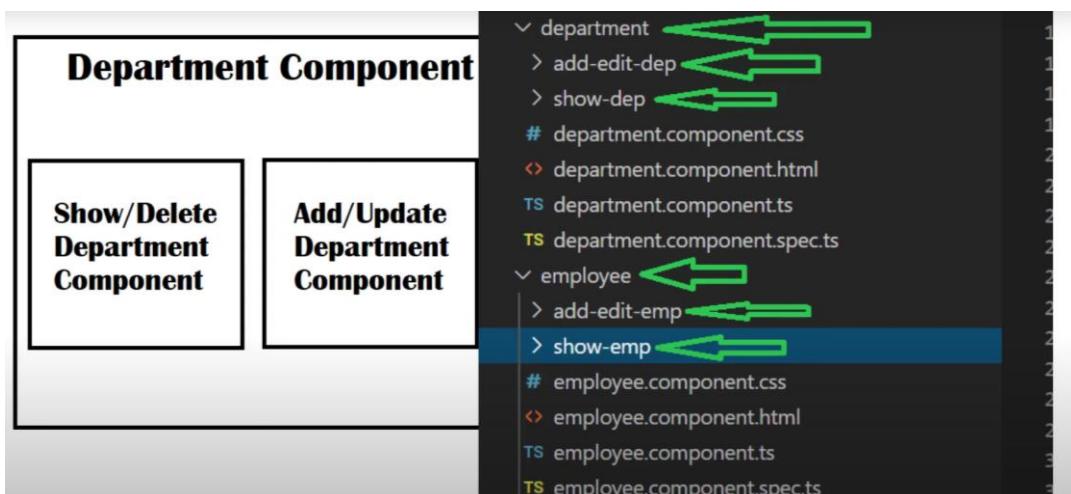
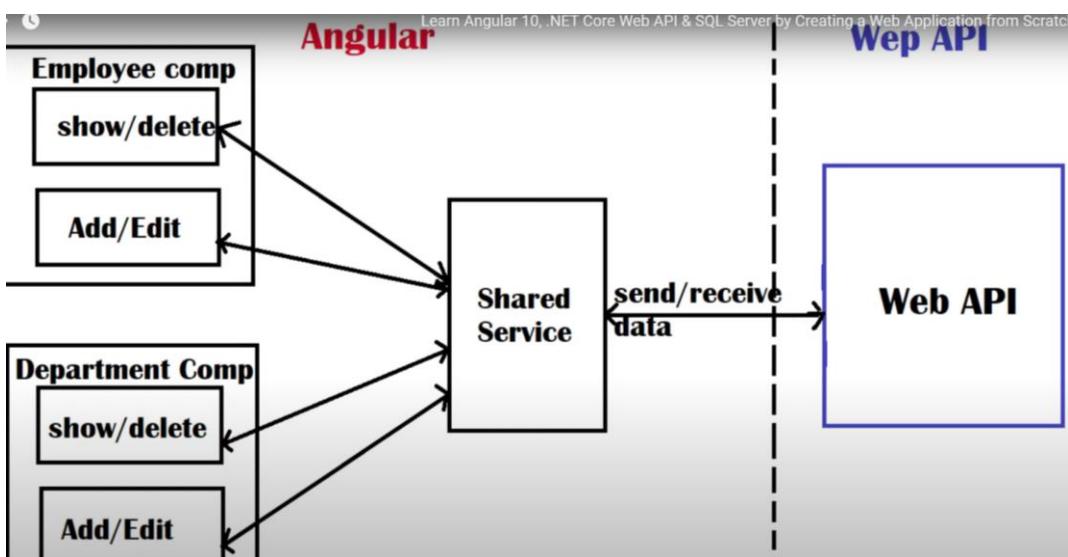
Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "DepartmentName": "WinGata"
4   },
5   {
6     "DepartmentName": "a"
7   },
8   {
9     "DepartmentName": "Support"
10}

```

200 OK 201 ms 230 B Save Response



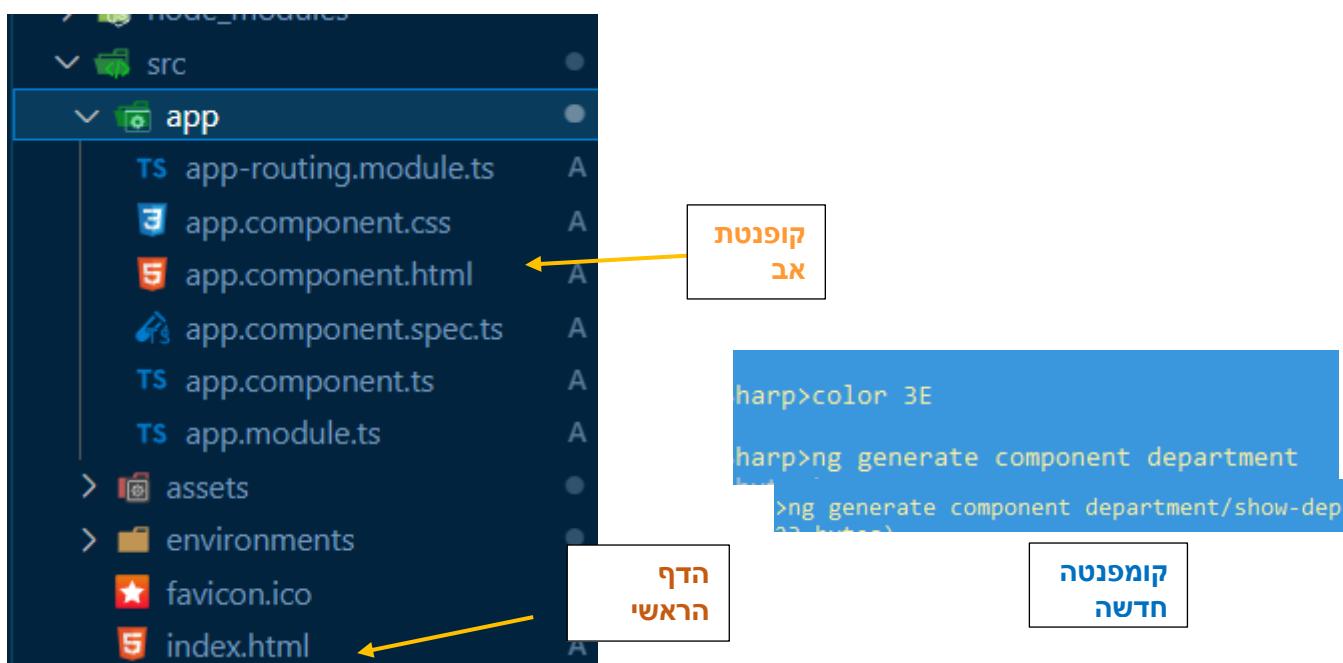
Schematic input does not validate against the Schema: {"projectRoot": "", "name": "first-AngularC##", "prefix": "app", "routg": true, "style": "css", "skipTests": false, "skipPackageJson": false, "skipInstall": true, "strict": true, "minimal": false}
Errors:

```
C:\Users\edard\Desktop\All Job\Angular>ng new first-Angular_CSharp
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
-CREATE first-Angular_CSharp/angular.json (3123 bytes)
CREATE first-Angular_CSharp/package.json (1083 bytes)
```

פרויקט אגולר חדש ב- רוח

```
\Users\edard>color 3E
\Users\edard>ng serve --open
```

פתיחת
האתר



```

c > app > TS app.component.ts > ...
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'first-Angular_C_Sharp';
10 }
11

```

ng generate component department

>ng generate service shared

שירות חדש

ng generate service serviceName

localhost:5140/swagger/index.html ⓘ ⌂ ⌄ ⌅ ⌆

אתר אינגולר

Ctrl + c -> exit
run angular

ng add @ng-bootstrap/ng-bootstrap

```
ts app.module.ts M
src > app > TS app.module.ts > ...
7 import { ShowDepComponent } from './department/show-
8 import { AddEditDepComponent } from './department/ad-
9 import { EmployeeComponent } from './employee/employ
10 import { ShowEmpComponent } from './employee/show-em
11 import { AddEditEmpComponent } from './employee/add-e
12 import {SharedService} from './shared.service'
13 ...
14 @NgModule({
15   declarations: [
16     AppComponent,
17     DepartmentComponent,
18     ShowDepComponent,
19     AddEditDepComponent,
20     EmployeeComponent,
21     ShowEmpComponent,
22     AddEditEmpComponent
23   ],
24   imports: [
25     BrowserModule,
26     AppRoutingModule
27   ],
28   providers: [SharedService],
29   bootstrap: [AppComponent]
30 })

```

ng add @ng-
bootstrap/ng-
bootstrap

import

import
{HttpClient}

HttpC

קריאה לקומפונטו

```
import {HttpClientModule} from '@angular/common/http';
import {FormsModule, ReactiveFormsModule} from '@angular/forms'
...
@NgModule({
  declarations: [
    AppComponent,
    DepartmentComponent,
    ShowDepComponent,
    AddEditDepComponent,
    EmployeeComponent,
    ShowEmpComponent,
    AddEditEmpComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [SharedService],

```

קריאה למטפל
form , http

```

-dep.component.html U   TS show-dep.component.ts U   TS shared.service.ts U X
p > TS shared.service.ts > SharedService
  providedIn: 'root'
}
export class SharedService {
  readonly APIUrl ="http://localhost:5140/api";
  readonly PhotoUrl ="http://localhost:5140/Photos";
  constructor(private http:HttpClient) { }
}

```

נדיר נתיב לבקשת api
ולכל תות תיקיה

readonly APIUrl
="http://localhost:5140/

import {HttpClient}
from

```

getDepList():Observable<any[]>{
  return this.http.get<any>(this.APIUrl+'/Department');
}
addDepartment(val:any){
  return this.http.post(this.APIUrl+'/Department',val);
}
updateDepartment(val:any){
  return this.http.put(this.APIUrl+'/Department',val);
}
deleteDepartment(val:any){
  return this.http.delete(this.APIUrl+'/Department/'+val);
}

getEmpList():Observable<any[]>{
  return this.http.get<any>(this.APIUrl+'/Employee');
}
addEmployee(val:any){
  return this.http.post(this.APIUrl+'/Employee',val);
}
updateEmployee(val:any){
  return this.http.put(this.APIUrl+'/Employee',val);
}
deleteEmployee(val:any){
  return this.http.delete(this.APIUrl+'/Employee/'+val);
}

UploadPhoto(val:any){
  return this.http.post(this.APIUrl+'/Employee/SaveFile',val)
}

getAllDepartmentName():Observable<any[]>{
  return this.http.get<any[]>(this.APIUrl+'/Employee/GetAllDepartmentNames')
}
}

```

API Request

getDepList():Ob
servable<any[]>{
 return
 this.http.get<any>
(this.APIUrl+'/D
epartment').

getEmpList():Ob
servable<any[]>{
 return
 this.http.get<any>
(this.APIUrl+'/Em
ployee').

UploadPhoto(val:
ny){
 return
 this.http.post(th
is.APIUrl+'/Emplo
yee/SaveFile',val)

הגדרת נתיבים
לאתר

```
shared.service.ts U  TS app-routing.module.ts A X
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { EmployeeComponent } from './employee/employee.component';
import { DepartmentComponent } from './department/department.component';

const routes: Routes = [
  {path: 'employee', component: EmployeeComponent},
  {path: 'department', component: DepartmentComponent}
];

```

הגדרת הצגת
הדף
המשתנים
לאתר

```
routing.module.ts M  TS app.component.html M X
margin-left: 72px;
}

svg#rocket-smoke {
  right: 120px;
  transform: rotate(-5deg);
}

@media screen and (max-width: 575px) {
  svg#rocket-smoke {
    display: none;
    visibility: hidden;
  }
}

<h2>Hello All Job !!</h2>
<router-outlet></router-outlet>
```

<link rel="stylesheet"
 href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"/>

<script
src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5NvP1z011zTO/6e9RHS/

<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-9/reFTGAW83EW2RDu2S2UpWVXVEIYFmS1PmM杨

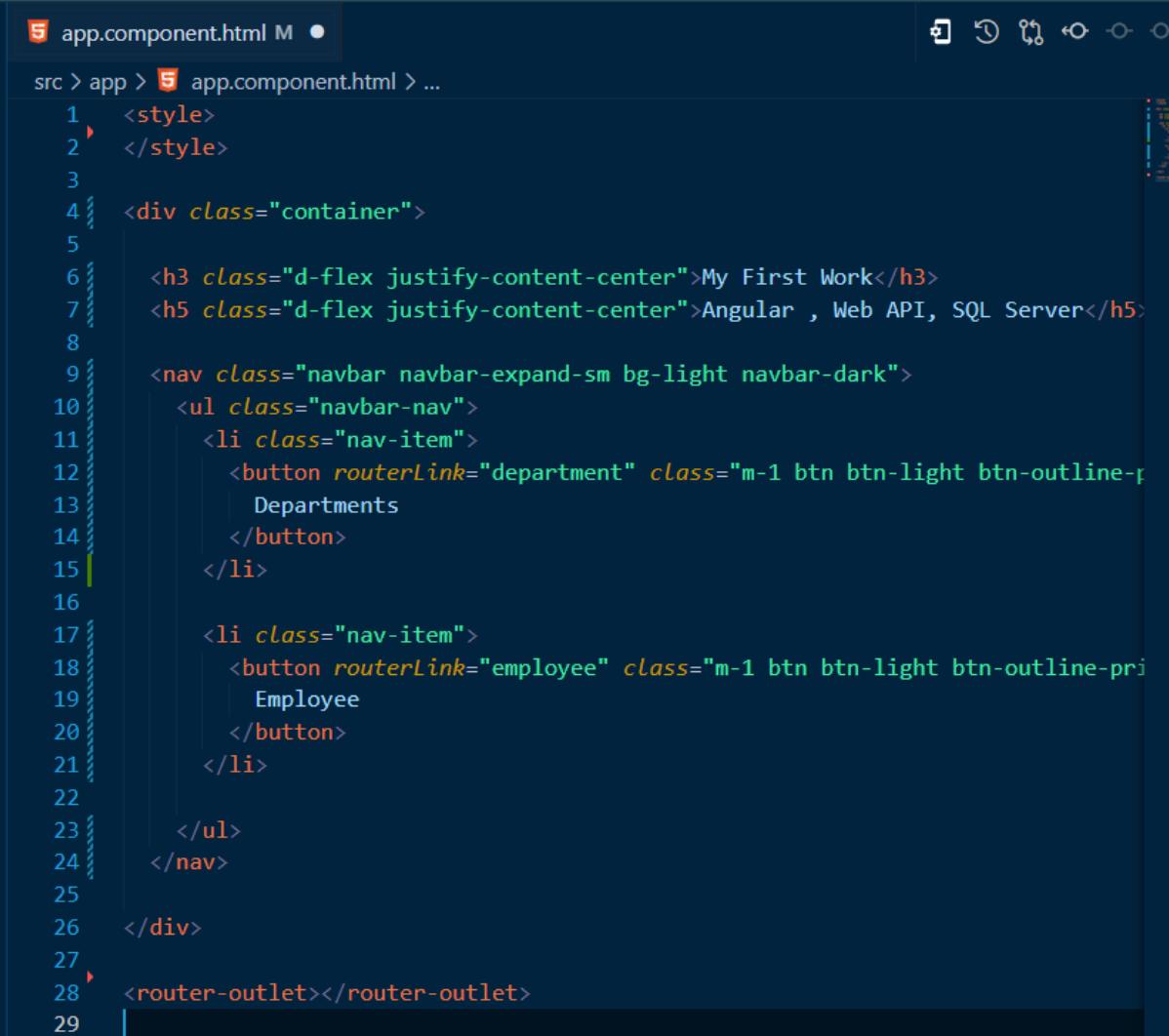
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" integrity="sha384-OgVRvuATPZyWZLjNtKwVjF0EiQ+uJ09f/FQ6fZStTTsF0sRZbB1n0Ql4

הגדרת עיצוב
bootstrap

```
c > TS index.html > ...
1  <!doctype html>
2  <html lang="en">
3  <head>
4  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
5  <meta charset="utf-8">
6  <title>FirstAngularCSharp</title>
7  <base href="/">
8  <meta name="viewport" content="width=device-width, initial-scale=1">
9  <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12 <app-root></app-root>
13 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5NvP1z011zTO/6e9RHS/

```

קומפוננטות הפעולות וההצגה



The screenshot shows a code editor window with the file `app.component.html` open. The code is written in HTML and includes some CSS classes. A specific line, number 29, is highlighted with a yellow box. The line contains the opening tag for a `div` element.

```
1 <style>
2 </style>
3
4 <div class="container">
5
6   <h3 class="d-flex justify-content-center">My First Work</h3>
7   <h5 class="d-flex justify-content-center">Angular , Web API, SQL Server</h5>
8
9   <nav class="navbar navbar-expand-sm bg-light navbar-dark">
10    <ul class="navbar-nav">
11      <li class="nav-item">
12        <button routerLink="department" class="m-1 btn btn-light btn-outline-pri
13          Departments
14        </button>
15      </li>
16
17      <li class="nav-item">
18        <button routerLink="employee" class="m-1 btn btn-light btn-outline-pri
19          Employee
20        </button>
21      </li>
22
23    </ul>
24  </nav>
25
26 </div>
27
28 <router-outlet></router-outlet>
29
```

קומפוננטה
ההצגה

```
show-dep.component.html U TS show-dep.component.ts U X
app > department > show-dep > TS show-dep.component.ts > ...
import { Component, OnInit } from '@angular/core';

import {SharedService} from 'src/app/shared.service';

@Component({
  selector: 'app-show-dep',
  templateUrl: './show-dep.component.html',
  styleUrls: ['./show-dep.component.css']
})
export class ShowDepComponent implements OnInit {

  constructor(private service:SharedService) { }

  DepartmentList:any = [];

  ngOnInit(): void {
    this.refreshDepList();
  }

  refreshDepList(){
    this.service.getDepList().subscribe(data=>{
      this.DepartmentList=data;
    })
  }
}
```

import
{SharedService}

constructor(priva
te
refreshDepList(){
this.service.

הקריאה אליה
תראה כך

```
department.component.html U
src > app > department > department.component.html
1 <app-show-dep></app-show-dep>
```

```
pp > department > show-dep > 5 show-dep.component.html > ...
<table class="table table-striped">
  <thead>
    <tr>
      <th>DepartmentId</th>
      <th>Department Name</th>
      <th>Options</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let dataItem of DepartmentList">
      <td>{{dataItem.DepartmentId}}</td>
      <td>{{dataItem.DepartmentName}}</td>
      <td>
        <button type="button" class="btn btn-light mr-1">
          Edit
        </button>
        <button type="button" class="btn btn-light mr-1">
          Delete
        </button>
      </td>
    </tr>
  </tbody>
</table>
```

הגדרת תצוגת המחלקה

הציג בולולאה את כל המחלקות

<table

| DepartmentId | Department Name | Options |
|--------------|-----------------|---|
| 1 | WinGata | <button>Edit</button> <button>Delete</button> |
| 5 | a | <button>Edit</button> <button>Delete</button> |
| 3 | Support | <button>Edit</button> <button>Delete</button> |

קומפוננטה העrica
של מחלקה

```

dep.component.html U ● TS add-edit-dep.component.ts U X ⌂ ⌂ ⌂ ⌂ ⌂
rc > app > department > add-edit-dep > TS add-edit-dep.component.ts > AddEditDepComponent
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-add-edit-dep',
5   templateUrl: './add-edit-dep.component.html',
6   styleUrls: ['./add-edit-dep.component.css']
7 })
8 export class AddEditDepComponent implements OnInit {
9
10   constructor() { }
11
12   @Input() dep:any;
13   DepartmentId:string="";
14   DepartmentName:string="";
15
16   ngOnInit(): void {
17     this.DepartmentId=this.dep.DepartmentId;
18     this.DepartmentName = this.dep.DepartmentName;
19   }
20 }

```

```

dep.component.ts U TS show-dep.component.ts U X
department > show-dep > TS show-dep.component.ts > ShowDepComponent
ModalTitle:string="";
ActivateAddEditDepComp:boolean=false;
dep:any;
myAddDep(){
  this.dep={
    DepartmentId:0,
    DepartmentName:"",
  }
  this.ModalTitle="Add Department";
  this.ActivateAddEditDepComp=true;
}
myCloseWindowModal(){
  this.ActivateAddEditDepComp=false;
  this.refreshDepList();
}

```

| |
|------------|
| ngOnInit() |
| ModalTitle |

הגדרת הוסףת
מחלקה

```

<!-- Button trigger modal -->
<!-- Cannot Close Window = data-backdrop+keyboard -->
<button type="button" class="btn btn-primary float-right m-2"
data-toggle="modal" data-target="#myModal"
(click)="myAddDep()"
data-backdrop="static" data-keyboard="false"
>

Add Department
</button>

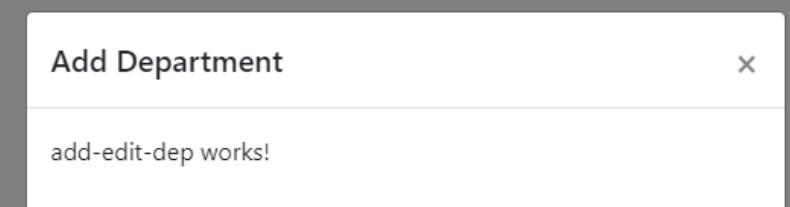
<!-- Modal -->
<div class="modal fade" id="myModal" tabindex="-1"
role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true"
ng-show="isPopupVisible">
<div class="modal-dialog modal-dialog-centered modal-xl" role="document">
    <div class="modal-content">
        <div class="modal-header">
            <h5 class="modal-title" id="exampleModalLabel">{{ModalTitle}}</h5>
            <button type="button" class="close"
data-dismiss="modal" aria-label="Close"
(click)="myCloseWindowModal()">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>
        <div class="modal-body">
            <app-add-edit-dep [dep]="dep" *ngIf="ActivateAddEditDepComp">
            </app-add-edit-dep>
        </div>
    </div>
</div>
</div>

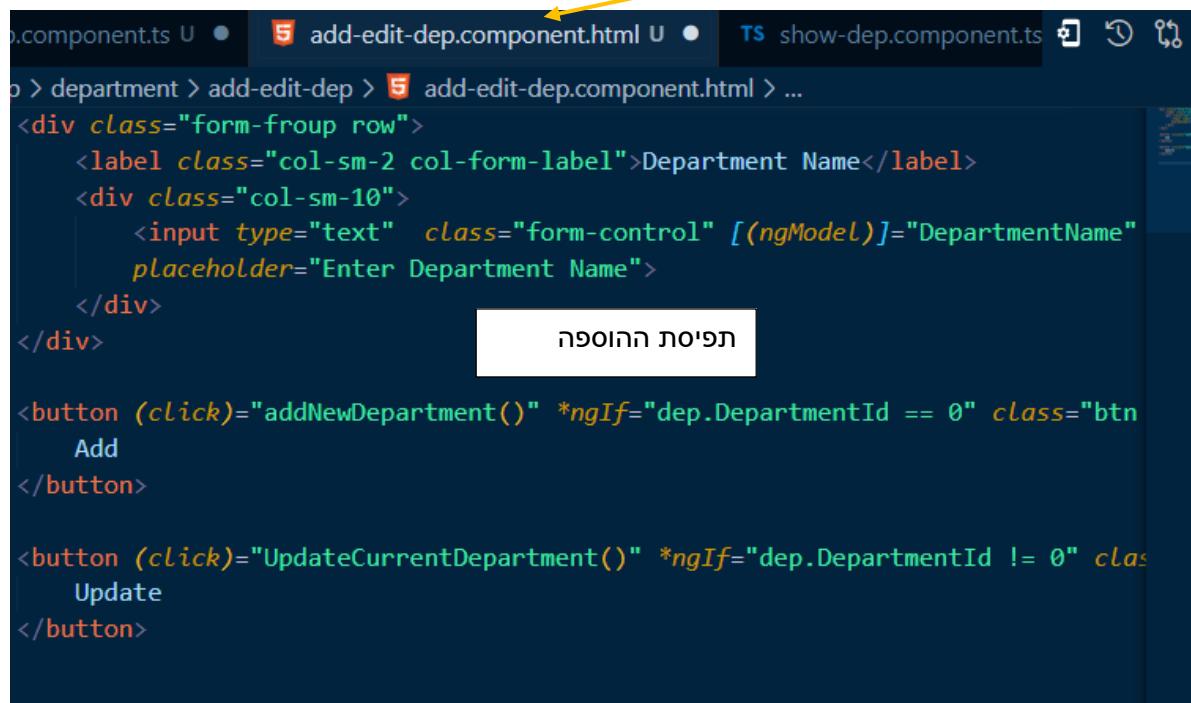
```

הגדרת פונקציית
ההוספה

<!-- Button -->
<!-- Modal -->
<div class="modal"

| My First Work | | |
|--|------------------------|---|
| Angular , Web API, SQL Server | | |
| Departments Employee | | |
| DepartmentId | Department Name | Options |
| 1 | WinGata | Edit Delete |
| 5 | a | Edit Delete |
| 3 | Support | Edit Delete |





add-edit-dep.component.html

```
<div class="form-froup row">
  <label class="col-sm-2 col-form-label">Department Name</label>
  <div class="col-sm-10">
    <input type="text" class="form-control" [(ngModel)]="DepartmentName"
      placeholder="Enter Department Name">
  </div>
</div>



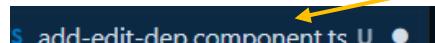
תפיסות ההויספה



<button (click)="addNewDepartment()" *ngIf="dep.DepartmentId == 0" class="btn-primary">
  Add
</button>

<button (click)="UpdateCurrentDepartment()" *ngIf="dep.DepartmentId != 0" class="btn-primary">
  Update
</button>
```

```
<div
  class="form-
  froup row">
```



```

src > app > department > add-edit-dep > TS add-edit-dep.component.ts > ...
1 import { Component, Input, OnInit } from '@angular/core';
2 import {SharedService} from 'src/app/shared.service'
3
4 @Component({
5   selector: 'app-add-edit-dep',
6   templateUrl: './add-edit-dep.component.html',
7   styleUrls: ['./add-edit-dep.component.css']
8 })
9 export class AddEditDepComponent implements OnInit {
10
11   constructor(private service:SharedService) { }
12
13   @Input() dep:any;
14   DepartmentId:string="";
15   DepartmentName:string="";
16
17
18   ngOnInit(): void {
19     this.DepartmentId=this.dep.DepartmentId;
20     this.DepartmentName = this.dep.DepartmentName;
21   }
22
23   addNewDepartment(){
24     var val = {DepartmentId:this.DepartmentId,
25               DepartmentName:this.DepartmentName};
26     this.service.addDepartment(val).subscribe(res=>{
27       alert(res.toString());
28     })
29   }
30
31   UpdateCurrentDepartment(){
32     var val = {DepartmentId:this.DepartmentId,
33               DepartmentName:this.DepartmentName};
34     this.service.updateDepartment(val).subscribe(res=>[
35       alert(res.toString())
36     ])
37   }
38 }
39

```

```

import
{SharedService
} from
constructor(p
rivate
service:Shared
service
addNewDepartmen
t(){
  var val =
UpdateCurrentDe
partment(){
  var val =

```

```

dep.component.html U • TS show-dep.component.ts U • ⌂ ⌂ ⌂ ⌂
> department > show-dep > TS show-dep.component.ts > ShowDepCom
refreshDepList(){
  this.service.getDepList().subscribe(data=>{
    this.DepartmentList=data;
  })
}

ModalTitle:string="";
ActivateAddEditDepComp:boolean=false;
dep:any;

myEditDep(item:any){
  this.dep=item;
  this.ModalTitle="Edit Department";
  this.ActivateAddEditDepComp=true;
}

myDeleteDep(item:any){
  if(confirm('Are you sure ??'))
  {
    this.service.deleteDepartment(item.DepartmentId).sub
    alert(data.toString());
    this.refreshDepList();
  }
}

```

```

myEditDep(item:any){
{
  this.dep=item;
  this.ModalTitle="Edit Department";
  this.ActivateAddEditDepComp=true;
}

myDeleteDep(item:any){
  if(confirm('Are
you sure ??'))
  {
    this.service.d

```



New in v1.8.0: 140+ new icons!

Bootstrap Icons

Free, high quality, open source icon library with over 1,600 icons.

Include them anyway you like—SVGs, SVG sprite, or web fonts.

Use them with or without [Bootstrap](#) in any project.

Copy HTML

Paste the SVG right into your project's code.

```

<svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor">
  <path d="M11.46.146A.5.5 0 0 0 11.107 .9H4.893a.5.5 0 0 0-.353.146L.146 4.54A.5.5 0
</svg>

```

```

<tbody>
  <tr *ngFor="let dataItem of DepartmentList">
    <td>{{dataItem.DepartmentId}}</td>
    <td>{{dataItem.DepartmentName}}</td>
    <td>
      <button type="button" class="btn btn-light mr-1" data-toggle="modal" data-target="#myModal"
        (click)="myEditDep(dataItem)" data-backdrop="static" data-keyboard="false">
        <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-pencil">
          <path fill-rule="evenodd" d="M10.646.646a.5.5 0 0 1 .708 .014 4a.5.5 0 0 1 0 .708l-.1902 1.902 1.902-.829 3.313a1.5 1.5 0 0 1 1.07 0h4.893a.5.5 0 0 0-.353.146L.146 4.54A.5.5 0 0 0 0 4.893z"/>
        </svg>
      </button>
      <button type="button" class="btn btn-light mr-1"
        (click)="myDeleteDep(dataItem)">
        <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor"
          class="bi bi-x-octagon-fill" viewBox="0 0 16 16">
          <path fill-rule="evenodd" d="M11.46.146a.5.5 0 0 0 11.107 0H4.893a.5.5 0 0 0-.353.146L.146 4.54A.5.5 0 0 0 0 4.893z"/>
        </svg>
      </button>
    </td>
  </tr>
</tbody>
</table>

```

הוספה כפטורית הוספה
ומחיקה לאלמנטי show

```

<tbody>
  <tr *ngFor="let
    dataItem of
    >

```



```
TS show-emp.component.ts U X
src > app > employee > show-emp > TS show-emp.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2
3 import {SharedService} from 'src/app/shared.service';
4
5
6 @Component({
7   selector: 'app-show-emp',
8   templateUrl: './show-emp.component.html',
9   styleUrls: ['./show-emp.component.css']
10 })
11 export class ShowEmpComponent implements OnInit {
12
13   constructor(private service:SharedService) { }
14
15   EmployeeList:any = [];
16
17   ModalTitle:string="";
18   ActivateAddEditEmpComp:boolean=false;
19   emp:any;
20
21
22   ngOnInit(): void {
23     this.refreshEmpList();
24   }
25
26   refreshEmpList(){
27     this.service.getEmpList().subscribe(data=>{
28       this.EmployeeList=data;
29     })
30   }
31
32   myEditEmp(item:any){
33     this.emp=item;
34     this.ModalTitle="Edit Employee";
35     this.ActivateAddEditEmpComp=true;
36   }
37
38   myDeleteEmp(item:any){
39     if(confirm('Are you sure ??'))
40     {
41       this.service.deleteEmployee(item.EmployeeId).subscribe(
42         data=>{
43           alert(data.toString());
44           this.refreshEmpList();
45         }
46     }
47   }
48
49   myAddEmp(){
50     this.emp={
51       EmployeeId:0,
52       EmployeeName:"",
53       Department:"",
54       DateOfJoining:"",
55       PhotoFileName:"anonymous.png",
56     }
57     this.ModalTitle="Add Employee";
58     this.ActivateAddEditEmpComp=true;
59   }
60
61   myCloseWindowModal(){
62     this.ActivateAddEditEmpComp=false;
63     this.refreshEmpList();
64   }
65 }
```

```
import
{SharedService}
from
constructor(priv
ate
service:SharedSe
rvice) { }

myEditEmp(item:a
ny){
  this.emp=ite
```

```
employee.component.html U X
src > app > employee > employee.component.html > ...
1 <app-show-emp></app-show-emp>
```

```

TS add-edit-emp.component.ts U ●
> app > employee > add-edit-emp > TS add-edit-emp.component.ts > AddEditEmpC
  1 import { Component, OnInit, Input } from '@angular/core';
  2
  3 import { SharedService } from 'src/app/shared.service'
  4
  5 @Component({
  6   selector: 'app-add-edit-emp',
  7   templateUrl: './add-edit-emp.component.html',
  8   styleUrls: ['./add-edit-emp.component.css']
  9 })
10 export class AddEditEmpComponent implements OnInit {
11
12   constructor(private service:SharedService) { }
13
14   @Input() emp:any;
15   EmployeeId:string="";
16   EmployeeName:string="";
17   Department:string="";
18   DateOfJoining:string="";
19   PhotoFileName:string="anonymous.png";
20
21   ngOnInit(): void {
22     this.EmployeeId=this.emp.EmployeeId;
23     this.EmployeeName = this.emp.EmployeeName;
24     this.Department = this.emp.Department;
25     this.DateOfJoining = this.emp.DateOfJoining;
26     this.PhotoFileName = this.emp.PhotoFileName;
27   }
28

```

```

import
{SharedService}
e} from
'src/app/shar
ed.service'

```

```

constructor
(private
service:Share
dService) { }

```

| EmployeeId | Employee Name | Department | DateOfJoining | Options |
|------------|---------------|------------|---------------|---|
| 1 | Sam | IT | 2020-06-01 |   |
| 3 | Tehila | MVC | 1998-12-13 |   |

```

TS add-edit-emp.component.ts U ●
src > app > employee > add-edit-emp > TS add-edit-emp.component.ts > ...
  20
  21   ngOnInit(): void {
  22     this.loadDepartmentList();
  23   }
  24
  25   PhotoFilePath:string="";

```

טיפול בתמונות

```

loadDepartmentList(){
  this.service.getAllDepartmentName().subscribe( (data:any)=>{
    this.DepartmentList = data;
  }

  this.EmployeeId=this.emp.EmployeeId;
  this.EmployeeName = this.emp.EmployeeName;
  this.Department = this.emp.Department;
  this.DateOfJoining = this.emp.DateOfJoining;
  this.PhotoFileName = this.emp.PhotoFileName;

  this.PhotoFilePath=this.service.PhotoUrl+"/"+this.PhotoFileName;
})
}

addNewEmployee(){
  var val = {EmployeeId:this.EmployeeId,
  EmployeeName:this.EmployeeName,
  Department:this.Department,
  DateOfJoining:this.DateOfJoining,
  PhotoFileName:this.PhotoFileName,
  };
  this.service.addEmployee(val).subscribe(res=>{
    alert(res.toString());
  })
}

UpdateCurrentEmployee(){
  var val = {EmployeeId:this.EmployeeId,
  EmployeeName:this.EmployeeName,
  Department:this.Department,
  DateOfJoining:this.DateOfJoining,
  PhotoFileName:this.PhotoFileName,
  };
  this.service.updateEmployee(val).subscribe(res=>{
    alert(res.toString());
  })
}

```

```

ngOnInit(): void {
  addNewEmployee();
}

UploadPhoto(event)

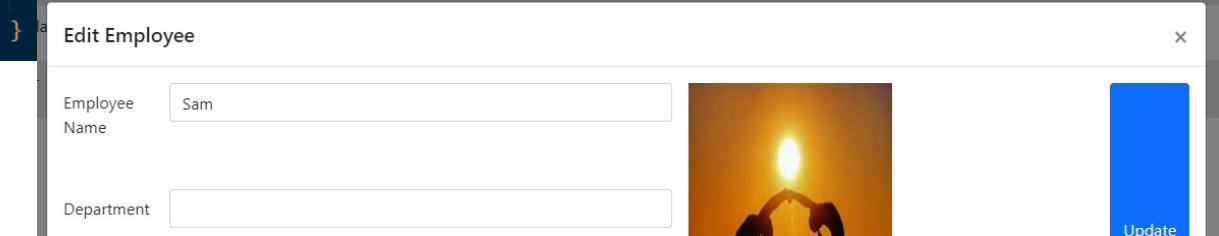
```

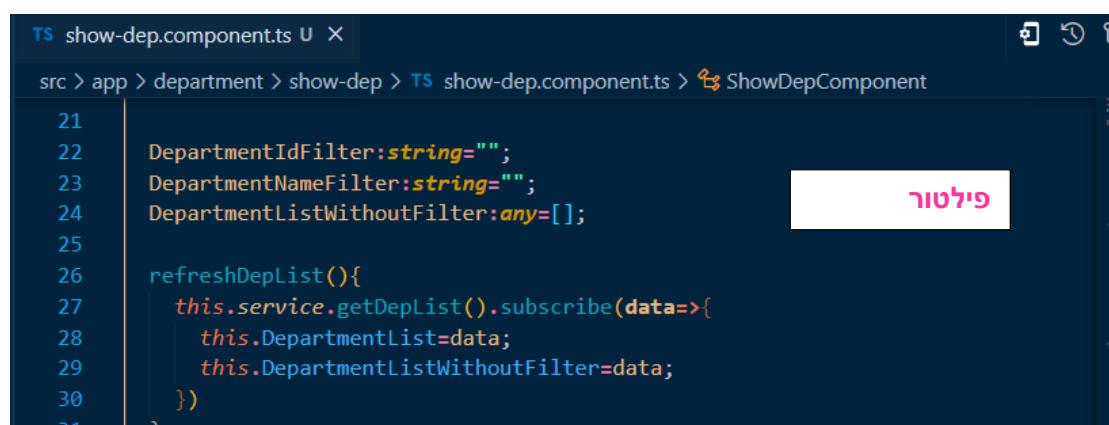
```

UploadPhoto(event:any){
  var file = event.target.files[0];
  const formData:FormData=new FormData();
  formData.append('uploadedFile',file,file.name);

  this.service.UploadPhoto(formData).subscribe( (data:any)=>{
    this.PhotoFileName = data.toString();
    this.PhotoFilePath = this.service.PhotoUrl+"/"+this.PhotoFileName;
  })
}

```





ts show-dep.component.ts u X

src > app > department > show-dep > ts show-dep.component.ts > ShowDepComponent

```
21 DepartmentIdFilter:string="";
22 DepartmentNameFilter:string="";
23 DepartmentListWithoutFilter:any=[];
24
25 refreshDepList(){
26   this.service.getDepList().subscribe(data=>{
27     this.DepartmentList=data;
28     this.DepartmentListWithoutFilter=data;
29   })
30 }
```

פִילטּוֹר

```
DepartmentI  
dFilter:str  
ing="";
```

```
<thead>  
    <tr>  
        <th>
```

```
dep.component.ts U show-dep.component.html X
show-dep.component.html > table.table-striped > tbody > tr >

<table class="table table-striped">

  <thead>
    <tr>
      <th>
        <div class="d-flex flex-row">
          <input [(ngModel)]="DepartmentIdFilter" class="form-control" (keyUp)="FilterFn" placeholder="Filter">
        </div>
        Department Id</th>
      <th>
        <div class="d-flex flex-row">
          <input [(ngModel)]="DepartmentNameFilter" class="form-control" (keyUp)="FilterFn" placeholder="Filter">
        </div>
        Department Name</th>
      <th>Options</th>
    </tr>

  sortResult(prop:string,asc:boolean){
    this.DepartmentList = this.DepartmentListWithoutFilter.sort(function(a:any,b:any){
      if(asc)
      {
        return(a[prop]>b[prop])? 1 : ((a[prop]<b[prop])) ? -1 : 0
      }

      else
      {
        return(b[prop]>a[prop])? 1 : ((b[prop]<a[prop])) ? -1 : 0
      }
    })
  }
}
```

Prop –
שם התוכנה
של האובייקט

```


show-dep.component.ts U show-dep.component.html U ●



> show-dep.component.html > table.table.table-striped > thead > tr > th > div.d-



<thead>



<tr>



<th>



<div class="d-flex flex-row">



<input [(ngModel)]="DepartmentIdFilter" class="form-control" (keyUp)="FilterFn()" placeholder="Filter"/>



<button type="button" class="btn btn-light" (click)="sortResult('DepartmentId',true)">



Up



</button>



<button type="button" class="btn btn-light" (click)="sortResult('DepartmentId',false)">



Down



</button>



</div>



DepartmentId</th>



<th>



<div class="d-flex flex-row">



<input [(ngModel)]="DepartmentNameFilter" class="form-control" (keyUp)="FilterFn()" placeholder="Filter"/>



<button type="button" class="btn btn-light" (click)="sortResult('DepartmentName',true)">



Up



</button>



<button type="button" class="btn btn-light" (click)="sortResult('DepartmentName',false)">



Down



</button>



</div>



Department Name</th>



<th>Options</th>



</tr>



</thead>


```

FilterFn(){
 var
 DepartmentIdFi
 lter =

<thead>
 <tr>

The (keyup) event is your best bet.

) Let's see why:

1. **(change)** like you mentioned triggers only when the input loses focus, hence is of limited use.
2. **(keypress)** triggers on key presses but doesn't trigger on certain keystrokes like the backspace.
3. **(keydown)** triggers every time a key is pushed down. Hence always lags by 1 character; as it gets the element state before the keystroke was registered.
4. **(keyup)** is your best bet as it triggers every time a key push event has completed, hence this also includes the most recent character.

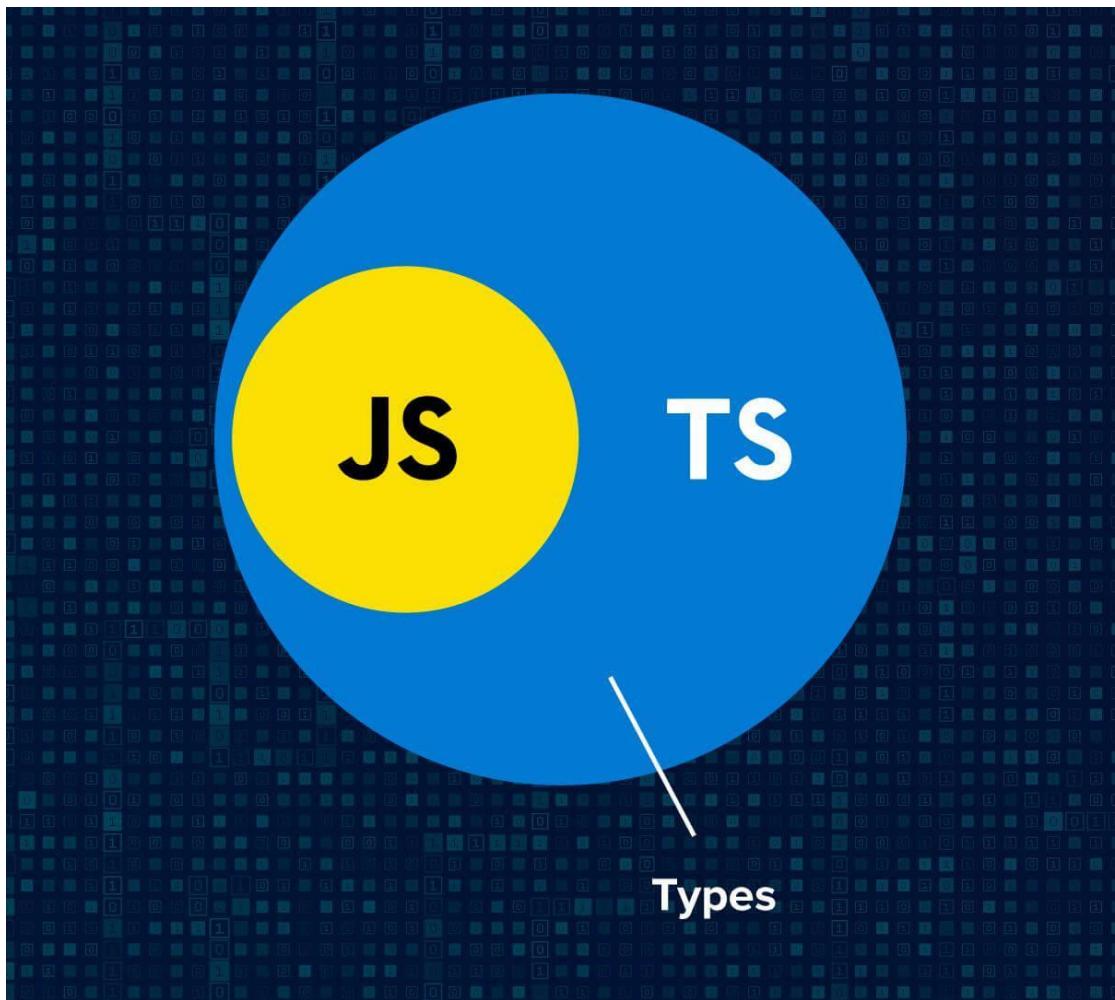
So (keyup) is the safest to go with because it...

- registers an event on every keystroke unlike (change) event
- includes the keys that (keypress) ignores
- has no lag unlike the (keydown) event

Share Follow

edited Apr 18, 2019 at 9:19

answered Sep 25, 2017 at 10:42



מבוא ל-typescript, ומה זה בכלל ?typescript

מחבר: יוסי בן הרוש בתאריך: 17.12.2016

היא סוג של javascript שמאפשר לשתמש בתכונות מונחה עצמים כבר היום ב-typescript. בין התכונות שהיא מושפה, static typing, C ו-PHP. תמייה במחלקות ותמייה במודולים ובדקוטורים.

הוא סופר-ט (superset) של javascript. הקוד מבוסס על javascript, וחייב לעבור קומpileaza (תרגום) ל-javascript כדי שהdeclaration ייבוטו מפני שדרוגים מביב javascript בלבד.

משתמשים ב-typescript מפני שהוא מוסיף ל-javascript מספר תכונות שלא קיימות בשפה, כפי שוראה בהמשך, אבל בעיקר מפני שכמעט כל התיעוד של .typescript כתוב ב-angular2.



אילו תכונות נוספו ל-typescript שלא קיימות ב-javascript?

typescript מוסיף ל-javascript יכולת לכתוב קוד בסגנון מונחה עצמים בדומה לשפות Java, C, PHP, C, Java. בין התכונות ניתן למנות את השימוש ב-, static typing, ארגון הקוד באמצעות מחלקות, מודולים ושימוש בדקוטורים.

נתחיל מהתכמה הראשונה static typing.

התכמה שנטנה ל-typescript את שמו היא static typing, שזה אומר שכךותבים קוד typescript צריך להגיד את סוג המשתנים. מקובל בשפות דוגמת C, Java, C, PHP. לדוגמה, אם המשתנה הוא מחרוזת אז נדרש להגיד לו אותו בהתאם, לדוגמה:

```
var name: string = "yossi";
```

שם המשתנה הוא name, ואת הסוג מצינים אחרי הנקודות (string).

תכמה נוספת היא התמייה במחלקות (class), בדומה לשפות אחרות של תכונות נוספות. מחלקות מאפשרות לקבץ קוד שישיר לנושא מסוים בתוך אותה מונחה עצמים. מחלקה, להחלטת אילו מתודות ותכונות של המחלקה להסתיר מפני הקוד שקיים מחוץ למחלקה, וגם אפשר שימוש חוזר בקוד, במחלקות אחרות, באמצעות הורשה.

התכמה השלישית מאפשרת את ארגון הקוד במודולים. זה מאפשר הוודת לכך שניתן לתת מספר תכונות או קבועים של קוד את אותו השם, את אותו -.namespace, ובכך להימנע ממצב של מחלקה אחת או יותר מפונקציה אחת יש אותו שם. הודות לשימוש במודולים ניתן להשתמש בביטחון בקוד שכתבנו מתכניםים שונים באותוuproject. ביל' צורך לריב על השמות של המחלקות. מה שמאפשר לנו לכתוב קוד גדול, גמיש ומתחכם הרבה יותר.

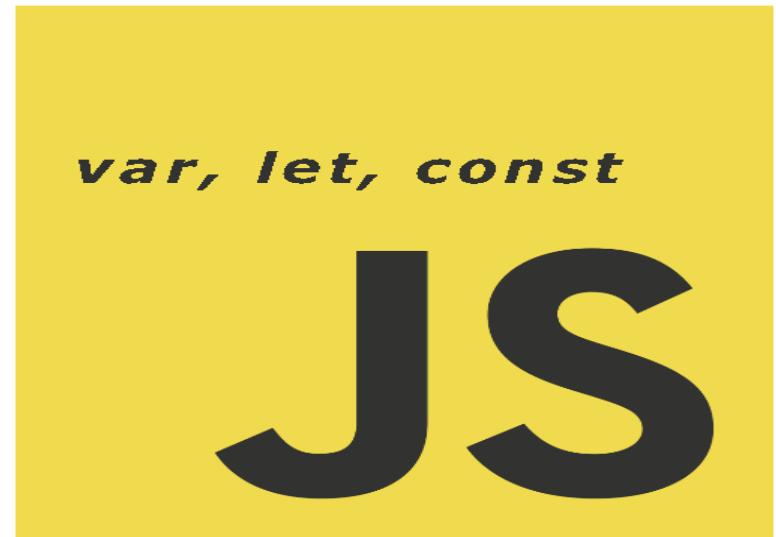
התכמה האחרונה ברשימה הוא השימוש בדקוטורים, שזה קוד שימושי יכולות למחלקות קיימות מבל' לשנות את קוד המחלקות עצמן.

- מבוא ל- [Typescript](#)
- [שלום עולם ב- Typescript](#)

הגדרת משתנים :

פעם, לפני שניים, הזכירתי להכריז על משתנים ב-javascript היינה באמצעות מילת המפתח `var`, אבל השימוש חלוף, והיום ניתן להשתמש ב-3 אפשרויות שונות לאוותה מטרה (או כמעט לאוותה מטרה).

ונתנו, איך לא, עם `var` הוותיק והחביב.



הכרזה על משתנים באמצעות `var`

אפשר להשתמש ב-`var` כדי להכריז על משתנים גם ב-typescript אבל הרבה מהקוד שכתב ב-typescript משתמש ב-`let` במקום ב-`var`.

זה מפני שהשימוש ב-`var` עלול לגרום להתנהגות בלתי צפויות.
לדוגמא:

* ואתם מוזמנים לנסות את התרגיל זהה, כמו את כל יתר התרגילים במדריך,
.chrome.בkonsole המפתחים של

```
if(x == 1){
    console.log("apple");
} else {
    console.log("bannana");
}

var x = 2; // bannana
```

התוצאה היא `bannana`, כלומר ש-`x` מוכח רק אחרי הבלוק של ה-`if`.

למה?

כ-`javascript` שמכריזים על משתנה באמצעות מילת המפתח `var` המשתנה מוחף בראש ה-scope שבו הוא מוגדר, ובמקרה שלנו לפני הבלוק של ה-`if`. זו התנהגות שמכונה `hoisting`, והיא מאפשרת לנו קודם להשתמש במשתנים, ורק אחר כך להכריז עליהם.

הכרזה על משתנים באמצעות מילת המפתח let

49

כדי לחסוך מאיינו התנהגויות מפתיעות מעין איליה הנטיה ב-javascript מודרני היא להזכיר על משתנים באמצעות **let**.

נכתב שוב את אותו הקוד, אבל נזכיר על המשתנה באמצעות מילת המפתח **let**, במקום באמצעות **var**.

```
if(x == 1){  
    console.log("apple");  
} else {  
    console.log("bannana");  
}  
  
let x = 2;
```

כשנريץ את הקוד, לדוגמה בקונסולת המפתחים של chrome, נקבל שגיאה כי אנחנו מנסים להשתמש ב-x, לפני שבכל הגדreno אותו.

עוד הבדל חשוב הוא ש-**let** קיים בرمת הבלוק שבתוכו הוא הוכרז, ולא בرمת ה-**scope**.

נכתב קוד עם **var**, ונזכיר על x בתוך בלוק של **for**:

```
for(var x=0; x<3; x++){  
    console.log(x);  
}  
  
console.log(x); // 3
```

אנו יכולים לגשת למשתנה x למורות שהוכרז בתוך הבלוק של ה-**for** גם מחוץ לבלוק. בגלל שימושים שהוכרזו עם **var** מוגדרים בرمת ה-**scope**, ולא בرمת הבלוק.

זה נכון גם לסוגים אחרים של בלוקים, דוגמת **לוואות** ו-**ife**.

לעומת זאת, אם נשתמש ב-**let** נקבל תוצאה שונה:

```
for(let x=0; x<3; x++){  
    console.log(x);  
}  
  
console.log(x); // Uncaught ReferenceError: x is not defined
```

והסיבה לשגיאה היא שימושה שהוכרז באמצעות **let** רק בתוך הבלוק שבתוכו הוכרז, ולא קיים מחוץ לבלוק.

לסיום, ניתן להזכיר על משתנים באמצעות **let** או באמצעות **var**, וישנו שני הבדלים בין משתנים שהוכרזו באמצעות **let** או **var**.

- הראשון, ש-**let** קיים רק בرمת הבלוק, ולא בرمת ה-**scope**.
- השני, השימוש ב-**var** מאפשר לנו להשתמש במשתנה עוד לפני שהוכרזו עלייו.

עכשו, הגיע הזמן להסביר תכונה נוספת שקיימת ב-javascript מודרני, והיא שנית להזכיר על קבועים.

הכרזה על קבועים באמצעות מילת המפתח `const`

כדי להכריז על קבועים נשתמש במילת המפתח `const`.

כמו בדוגמה:

```
const MYNAMEIS = "yossi";
```

את שמו של הקבוע נקבע נקפיד לכתוב באותיות גדולות.

אם עשינו ננסה להציב ערך אחר לתוך הקבוע, נקבל שגיאה.

```
MYNAMEIS = "Slim Shady"; // Uncaught TypeError:  
Assignment to constant variable.
```

זה הבדל העיקרי ממשתנה, והסיבה שבגללה קבוע הוא קבוע.
כי אסור שערך של קבוע ישתנה.

אפשר גם להציב אובייקט בתור ערך של קבוע, לדוגמה:

```
const PERSON = {name: "yossi", occupation: "webdev"};
```

ועכשיו, אם ננסה לדרוס את האובייקט כלו נקבל שגיאה:

```
const PERSON = null; // Uncaught SyntaxError: Identifier  
'PERSON' has already been declared
```

סיבת השגיאה ברורה, והוא נובעת מזה שאנחנו מנסים לשנות את ערכו של קבוע.

אבל פה צריך להזהר!!

למרות שלא ניתן לדרוס את כל האובייקט, עדין ניתן לשנות את הערכים שמוצבים לשוחות של האובייקט, זהה למקרה שהכרזנו עליו בתור קבוע.

לדוגמה, נמשיך עם אותו האובייקט, PERSON, ונגדיר ערכים חדשים לשודות:

```
PERSON.name = "Eminem";  
PERSON.occupation = "rapper";
```

וכשרץ בקונסולת המפתחים של chrome נגלה שאכן הצלחנו להגדיר מחדש את הערכים שמוצבים לשודות:

```
console.log(PERSON.name + " is a "  
+PERSON.occupation); // Eminem is a rapper
```

מדריך זה הוא המדריך הרביעי בסדרת המדריכים בנושא **TypeScript**. המדריכים הקיימים בסידרה הם:

- [מבוא ל- TypeScript](#)
- [שלום עולם ב- TypeScript](#)
- [var, let, const – javascript מודרני](#)

(**types**), כפי שמעיד השם, מצוי באפשרות להגדיר את סוג המשתנים (**types**) שמשמשים בקוד. במדריך זה נכיר שתי דרכי עיקריות להגדיר את סוג המשתנים, ונלמד מה כדאי לעשותו, וממה עדיף להימנע שימושים על משתנים.

דרך ראשונה: הגדרת סוג המשתנה בשורה שבה מרכיבים על המשתנה

הדרך הראשונה להגדיר את סוג המשתנה היאiacabar באוטה השורה שבה אנו מרכיבים על המשתנה.

לדוגמה, נכתוב את הקוד הבא בקובץ שמו `: types.ts`:

```
let myNumber : number;
myNumber = 42;
```

בשורה הראשונה, הגדרנו שהמשתנה `myNumber` שייר לסוג `number`, וכן יכול לקבל ערכים מספוריים בלבד (קראו על יתרונות השימוש ב-`let` במקום ב-`var` כשמרכיבים על משתנים). בשורה השנייה הצבעו את ערכו של המשתנה (42).

מכיוון שהגדרנו שהמשתנה חיב לקבל ערך מספרי בלבד, ניסו להציב ערך שאינו מסpter, יגרום לשגיאה בקומpileציה.

```
let myNumber : number;
myNumber = "forty two";
```

נקمل את הקובץ על ידי הריצת הפקודה הבאה על הקובץ `ts`, שבתוכו כתבנו את הקוד, בשורת הפקודות או בטרמינל (קראו כיצד לעשות קומpileציה של `: (typescript`)

```
> tsc types.ts
```

הקומpileר ייתן לנו הודעה שגיאה. לדוגמה:

```
> types.ts(2,1): error TS2322: Type "forty two" is not assignable to type 'number'.
```

אומנם הקומpileר הראה שגיאה, אבל הקובץ `javascript`, שיתקבל כתוצאה מהקומpileציה, ייתן קוד תקין שעבוד ללא בעיה. והסיבה היא שהקובץ מLETECODE – javascript, וב-`javascript` אין סוגים משתנים חזקים, ולכן ניתן לשנות את סוג המשתנה ללא בעיות.

ומזה אנחנו יכולים להסיק, שיכולים להיות מינים שונים שבهم קוד `typescript`, שאינו תקין, יכול להיות מLETECODE – javascript תקין, שעבוד ללא בעיות.

הדרך השנייה להגדיר את סוג המשתנה היא על ידי הצבת ערך למשתנה כבר בשורה הראשונה שבה מכוידים על המשתנה.

לדוגמה:

```
let myNumber = 42;
```

ועכשיו, אם ננסה להציב ערך שאינו מספר, הקומפיילר של typescript יראה שגיאה.

```
let myNumber = 42;
myNumber = "forty two";
```

כך נראהית השגיאה בהרצה שעשית על המחשב עצמו.

```
> types.ts(2,1): error TS2322: Type "forty two" is not assignable to type 'number'.
```

השגיאה בקומפיילציה נובעת מזה שtypescript הסיק מהשורה שבה הוכחה המשתנה לראשונה שהוא סוג המשתנה הוא מספר, וכך ניתן להציב ערך שאינו מספר לתוכו.

שים לב,.typescript מסוגל להסיק את סוג המשתנה אם מיד כמשמעותם על המשתנה, גם מציבים לו ערך, וזאת, ללא צורך לציין את סוג המשתנה.

כפי שראינו, ניתן להזכיר ערך משתנים בלי לציין באופן מפורש את הסוג, אבל עדיף להקפיד להגדיר את סוג המשתנים כדי להפוך את הקוד לקריא יותר, להפחית בשגיאות, ולקבל הודעה שגיאה אינפורטטיבית יותר מה-IDE! לפיכך, תמיד מומלץ להגדיר באופן מפורש את סוג המשתנה בשורה שבה אנו מכוידים עליו לראשונה.

לדוגמה:

```
let myNumber: number = 42;
```

ומה יקרה אם לא נקבעת תקינה להגדיר את סוג המשתנים?

אם לא נקבעת תקינה להגדיר את סוג המשתנים, אנחנו עלולים לשגות, ולהציב ערך ששייך לסוג לא נכון.

לדוגמה, הקוד הבא הוא תקין לגמרי, אבל אינו מומלץ בעילו.

```
let myNumber;
myNumber = 42;
myNumber = "forty two";
```

בשורה הראשונה הכרזנו על המשתנה בלי להגדיר את סוגו, בשורה השנייה החלטנו לו ערך של מספר, ובשורה השלישית החלטנו לו אותו המשתנה ערך של מחרוזת. זה ביל' לגרום לשגיאת קומפיילציה של typescript.

והסבירה לכך שלמשתנה שמכורץ בלי להגדיר את סוגו ניתן להציב אחר כך כל סוג של ערך היא שבמידה ולא מגדירים את סוג המשתנה,typescript מסיק שהסוג הוא.any

```
let myNumber: any;
myNumber = 42;
myNumber = "forty two";
```

זה כמובן מפספס את הפואנטה של שימוש ב-`typescript` כדי למנוע הצבת סוג לא נכון של ערכים, וכן נשתדל לצמצם את השימוש בסוג `any` (שמאפשר הצבת כל סוג של ערכים) עד כמה אפשר.

אילו סוגי בסיסיים קיימים ב- `TypeScript` ?

הסוג מחרוזת (`string`)

```
let myString: string = "this is a string";
```

הסוג מספר (`number`)

```
let myNumber: number = 42;
```

הסוג `boolean`

```
let myBoolean: boolean = true;
```

הסוג `any` (שמאפשר הצבה של כל סוג ערך).

```
let anything: any;
```

מערכות ב-TypeScript

מחבר: יוסי בן הרוש בתאריך: 26.01.2017

אחרי שבמדריך קודם למדנו על סוגי משתנים בסיסיים ב-[typescript](#), במדריך זה נלמד על מערכים.

מערכות ב-[typescript](#), מאפשרים להחזיק מספר ערכים תחת שם משתנה אחד.

דרך ראשונה להכיר על מערכת, היא להזכיר את שם המערכת ואת סוג הנתונים שהמערכת צריכה להכיל בשורה אחת, ולהציב את הערכים בהמשך.

```
let digits: number[];  
digits = [1,3,5,7];
```

כשמזכירם על מערכת, כדאי להגיד את סוג המשתנים שאנחנו מעוניינים שהוא יכול. במקרה זה, סוג המשתנים הוא `number`.

אפשר גם להציב את הערכים כבר באותה השורה שבה מזכירם על המערכת.

```
let letters: string[] = ["a", "b", "c"];
```

במידה ומצביעים את הערכים באותה השורה שבה מזכירם על המערכת, [typescript](#) יכול להסיק את סוג המשתנים גם בלי שנזכיר עליו.

במידה ונרצה להציב יותר מסוג אחד של ערכים במערכת, אפשר לעשות זאת באמצעות הסוג `any`.

```
let alphaNumerics: any[] = ["a", "b", "c", 1, 2, 3];
```

ניתן לגשת לפריטים במערכת כמו שניגשים לפריטים במערכת `javascript` רגיל.

```
let first:string[] = alphaNumerics[0];  
console.log(first); // "a"
```

וגם כן ב-[javascript](#) ניתן לגשת לכל אחד מהפריטים במערכת בתוך לולאת `.for...in`

```
for(let item in alphaNumerics){  
    console.log(item); // "0", "1", "2", "3", "4", "5"  
}
```

וכאן המქום לשאול איך נחזיר את הערכים של המערך במקום את המפתחות.

הדרך המקובלת ב-JavaScript היא הדרך הבאה.

```
for(let i in alphaNumerics){  
    console.log(alphaNumerics[i]); // "a", "b", "c", 1, 2, 3  
}
```

ואפשר גם באמצעות **of...of**typescript **for...of** מוציא לנו כדי לגשת ישירות לערכים של המשתנה.

```
for(let item of alphaNumerics){  
    console.log(item); // "a", "b", "c", 1, 2, 3  
}
```

פונקציות שלtypescript לטיפול במערכות

מציע שפע של פונקציות לטיפול במערכות. בוא נכיר כמה מהם.

concat()

מחבר שני מערכים לערך אחד.

```
let letters: string[] = ["a", "b", "c"];
let digits: string[] = ["1", "2", "3"];

let alphanumerics: string[] =
letters.concat(digits);
console.log(alphanumerics); //["a", "b", "c", "1", "2", "3"]
```

join()

מחבר את פריטי המערך למחרוזת אחת.

```
let str: string = alphanumerics.join();
console.log(str); //a,b,c,1,2,3
```

אם לא מציינים במפורש מה יחבר בין הפריטים, typescript מוסיף פסיקן.

וניתן גם לציג באופן מפורש מה יחבר בין הפריטים של המערך.

```
let str1: string = alphanumerics.join("|");
console.log(str1); //a | b | c | 1 | 2 | 3
```

map()

מחזיר מערך שבו כל פריט הוא תוצאה של קריאה לפונקציה מסוימת על כל פריט של המערך המקורי. בדוגמה הבאה, נוציא שורש ריבועי מכל אחד מהפריטים במערך.

```
let numbers: number[] = [1,9,25];
let roots: number[] = numbers.map(Math.sqrt);
console.log(roots); // [1, 3, 5]
```

push()

אפשר להוסיף פריט אחד, או מספר פריטים לסוף של מערך, ומוחזיר את מספר הפריטים במערך.

```
let numbers: number[] = [1,3,5];
```

נוסיף פריט אחד.

```
numbers.push(7);
console.log(numbers); // [1, 3, 5, 7]
```

push מוחזיר את מספר הפריטים במערך.

```
let len: number = numbers.push(15);
console.log(len); // 5
```

sort()

מסדר את הפריטים במערך.

```
let favFoods: string[] = ["falafel", "kabab", "amba"];
console.log(favFoods.sort()); // ["amba", "falafel", "kabab"]
```

toString()

מחזיר מחזורת שמייצגת את המערך.

```
let s: string = favFoods.toString();
console.log(s); // falafel,kabab,amba
```

ניתן לשלב בין מספר פונקציות כדי לקבל את התוצאה הרצויה. לדוגמה:

```
let s: string = favFoods.sort().toString();
console.log(s); // falafel,kabab,amba
```

Array.prototype.splice()

The `splice()` method changes the contents of an array by removing or replacing existing elements and/or adding new elements [in place](#). To access part of an array without modifying it, see [slice\(\)](#).

Try it

JavaScript Demo: Array.splice()

```

1 const months = ['Jan', 'March', 'April', 'June'];
2 months.splice(1, 0, 'Feb');
3 // inserts at index 1
4 console.log(months);
5 // expected output: Array ["Jan", "Feb", "March", "April", "June"]
6
7 months.splice(4, 1, 'May');
8 // replaces 1 element at index 4
9 console.log(months);
10 // expected output: Array ["Jan", "Feb", "March", "April", "May"]
11

```

String.prototype.split()

The `split()` method divides a [String](#) into an ordered list of substrings, puts these substrings into an array, and returns the array. The division is done by searching for a pattern; where the pattern is provided as the first parameter in the method's call.

Try it

JavaScript Demo: String.split()

```

1 const str = 'The quick brown fox jumps over the lazy dog.';
2
3 const words = str.split(' ');
4 console.log(words[3]);
5 // expected output: "fox"
6
7 const chars = str.split('');
8 console.log(chars[8]);
9 // expected output: "k"
10
11 const strCopy = str.split();
12 console.log(strCopy);
13 // expected output: Array ["The quick brown fox jumps over the lazy dog."]
14

```

```

1 import React from "react";
2
3 import { stringifyImageSizes } from "./utils";
4
5 const Images = () => {
6   const imageSizes = [
7     { name: "horizontal", width: 600, height: 380 },
8     { name: "vertical", width: 400, height: 650 },
9     { name: "thumbnail", width: 300, height: 300 },
10   ];
11
12   const normalizedImageStrings = stringifyImageSizes(imageSizes);
13
14   return (
15     <div className="images">
16       {normalizedImageStrings.map((s) => (
17         <div className="image-type">{s}</div>
18       ))}
19     </div>
20   );
21 };

```

Array.prototype.slice()

The `slice()` method returns a [shallow copy](#) of a portion of an array into a new array object selected from `start` to `end` (`end` not included) where `start` and `end` represent the index of items in that array. The original array will not be modified.

Try it

JavaScript Demo: Array.slice()

```

1 const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
2
3 console.log(animals.slice(2));
4 // expected output: Array ["camel", "duck", "elephant"]
5
6 console.log(animals.slice(2, 4));
7 // expected output: Array ["camel", "duck"]
8
9 console.log(animals.slice(1, 5));
10 // expected output: Array ["bison", "camel", "duck", "elephant"]
11
12 console.log(animals.slice(-2));
13 // expected output: Array ["duck", "elephant"]
14
15 console.log(animals.slice(2, -1));
16 // expected output: Array ["camel", "duck"]
17
18 console.log(animals.slice());
19 // expected output: Array ["ant", "bison", "camel", "duck", "elephant"]
20

```

Array.prototype.filter()

The `filter()` method **creates a new array** with all elements that pass the test implemented by the provided function.

Try it

JavaScript Demo: Array.filter()

```

1 const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
2
3 const result = words.filter(word => word.length > 6);
4
5 console.log(result);
6 // expected output: Array ["exuberant", "destruction", "present"]
7

```

פונקציות ב-TypeScript

מחבר: יוסי בן הרוש בתאריך: 03.02.2017

נית להציג על פונקציה ב-JavaScript בדרך הבאה.

```
function calcSum(x, y){  
    return x+y;  
}
```

מוסיף לנו את האפשרות להגיד את סוג המשתנים בפונקציה:

א. סוג המשתנים של הפרמטרים.

ב. סוג המשתנים שהוחזרה הפונקציה.

```
function calcSum(x:number, y:number):number {  
    return x+y;  
}
```

נית להוסיף את סוג המשתנה לכל אחד מהפרמטרים שמעברים לפונקציה.

בדוגמה זו, הפרמטרים x ו-y שייכים לסוג number.

נית להגיד את סוג המשתנה שאנו מעוניינים שהפונקציה תחזיר.

בדוגמה לעיל, הסוג המוחזר שייך לסוג number.

הסוג המוחזר void

כשהפונקציה לא מחזירה שום דבר הסוג המוחזר הוא void.
void שימושי בעיקר כשהפונקציה מדפסת למסך או לკונסולה במקום להחזיר ערך
באמצעות return.

בדוגמה הבאה, הפונקציה מדפסה ערך למסך ואינה מחזירה דבר, ולכן הסוג
המוחזר מהפונקציה הוא void

```
function writeSomething(something:any):void {  
    document.write(something);  
}
```

באופן רגילים, מספר הפרמטרים שמעוברים לפונקציה צריך להתאים למספר הפרמטרים שהפונקציה מצפה לקבל.

```
function calcSum(x:number, y:number):number {
    return x+y;
}

שגיית קומפיילציה כי הפונקציה מצפה לשני פרמטרים והעברם רק אחד
שגיית קומפיילציה כי הפונקציה מצפה לשני פרמטרים והעברם שלושה
let sum3 = calcSum(1,2); //3
```

כדי להפוך פרמטר לאופציוני, נוסיף סימן שאלה (?) אחרי שם הפרמטר. לדוגמה, אנחנו מעוניינים שהפרמטר `y` יהיה אופציוני, ולכן נוסיף אליו "?".

```
function calcSum(x:number, y?:number):number {
    if(!y) return x;

    return x+y;
}

let sum1 = calcSum(1); //1
let sum2 = calcSum(1,2); //3

שגיית קומפיילציה כי הפונקציה מצפה לשני פרמטרים והעברם שלושה
```

שים לבו את הפרמטרים האופציונליים חוברים מקום בסוף רשימת הפרמטרים. רק אחרי כל הפרמטרים שאינם אופציונליים.

פרמטרים בירית מחדל

ניתן להגדיר פרמטרים בירית מחדל, שיישמשו במקרה הפרמטרים שלא יועברו לפונקציה. בדוגמה הבאה, הערך בירית המחדל של `y` הוא 0.

```
function calcSum(x:number, y:number=0):number {
    return x+y;
}

let sum1 = calcSum(1); //1
let sum2 = calcSum(1,2); //3
```

גם אם נעביר לפמטר `y` ערך שאינו מוגדר, הערך בירית המחדל יתפוך את מקומו.

```
let sum3 = calcSum(1,undefined); //1
```

rest parameters

פרמטרים מסוג **rest** משמשים כشرطים לעבוד עם מספר פרמטרים, או שאין ידועים מה מספר הפרמטרים הצפוי שצורך להעיבר לפונקציה.

כדי להגדיר פרמטר מסוג **rest**, מוסיפים לפני שם המשתנה שלוש נקודות (...), וווג המשתנה הוא בהכרח מערך.

דוגמה הבאה, `restOfNums` הוא פרמטר מסוג **rest**. לכן, לפני השם הוספנו 3 נקודות, וווג הפרמטר הוא מערך.
בנוסף, ישנו שני פרמטרים נוספים `fNum` ו-`sNum`.

```
function calcSum(fNum: number, sNum: number,
...restOfNums: number[]):number {

    var sum = 0;
    for(let n of restOfNums){
        sum+=n;
    }

    return fNum + sNum + sum;
}
```

בעוד חשוב להזכיר את הפרמטרים הבאים אופציוניים, אפשר לבחור שלא להעיבר את הפרמטרים מסוג **rest**.
דוגמה הבאה, נעביר את 2 הפרמטרים הראשונים כאינם אופציוניים, ולא נעביר את הפרמטר השלישי כי הוא פרמטר **rest**, ולכן אופציוני.

```
let sum1 = calcSum(1,2); //3
```

דוגמה הבאה, מלבד שני הערךים הראשוניים, הלא אופציוניים, נעביר 3 ערכים נוספים ל-`restOfNums`, שיכל לקבל פרמטר אחד, שני פרמטרים או כמה שצורך כי הוא פרמטר **rest**. במקרה זה נעביר לו שלושה ערכים (3,4,5).

```
let sum2 = calcSum(1,2, 3, 4, 5); //15
```

ח-יה סדורה, או בשמה העברי הלא רשמי – רשיימה, הינה מערך שבו אנו מוגבלים מראש את סוג האיברים שהמערך יוכל להכיל.

```
1 | var x: [string, number] = ["hello", 10];
```

יש לנו לב שסדר השמת הסוגיםמשמעותי:

```
1 | var x: [string, number] = [0, "hello"] // error: Type '[number, string]' is not assignable
2 |
3 | var x: [string, number] = ["hello", 10] // valid
```

בניגוד לשפות אחרות כמו Python שבהן לא ניתן לעדכן את הרשימה לאחר הגדרתה, ב – TypeScript ניתן להוסיף איברים נוספים, כל עוד הם נכללים ברשימה הסוגים שהוגדרו:

```
1 | var x: [string, number] = ["hello", 10];
2 |
3 |
4 | x.push(20); // valid - type number
5 | x.push("world"); // valid - type string
6 | x.push(true); // invalid - type boolean
```

enum (רשימתבחירה)

סוג נתונים המאפשר לתת לקבוצת ערכים מסוימים שמות "ידידותיים". המהדר נותן לכל איבר ערך מסוים, כשהאיבר הראשון מקבל 0:

```
1 | enum Color {Red, White, Blue};
2 |
3 |
4 | var c: Color = Color.Blue; // 2
```

TypeScript מאפשרת למפתח לקבוע בעצמו את הערך שייקבל כל איבר:

```
1 | enum Color {Red = 4, White = 0, Blue = 9};
2 |
3 |
4 | var c: Color = Color.Blue; // 9
```

אפשרות נוספת נוספת היא לקבל את שם האיבר לפי הערך המספריו שלו:

```
1 | enum Color {Red, White, Blue};
2 |
3 |
4 | var c: string = Color[2]; // Blue
```

בחלק השני של המדריך, ארכיב על אפשרות נוספת שסוג הנתונים מכיל.

הצמדת סוג נתונים (type assertion)

TypeScript מאפשרת לעקוף את מנגנון זהה סוג הנתונים של המהדר, ולהתיחס לסוג נתונים מסוים כסוג נתונים אחר, בשביל להתמודד עם מצב שבו המהדר קובע שהשמה שבוצעה לא תקינה, אך אתה כמתכנת יודע שמדובר במקרה מיוחד ותיק.

הדוגמה הבאה ממחישה מצב שכזה:

```

1  interface Teacher {
2    name: string,
3    birthdate: Date,
4    profession: string
5  }
6
7  interface Principle extends Teacher {
8    startDate: Date
9  }
10
11 var t: Teacher = {
12   name: "Israel Israeli",
13   birthdate: new Date("1966-04-04"),
14   profession: "Mathematics"
15 };
16
17
18

```

קטע הקוד מתאר שני ממשקים: מורה ומנהל, שיורש ממשיק המורה. בנוסף, יצירתו משתנה שמכיל אובייקט מסווג מורה.

עת, ישראל (המורה), קודם לתפקיד מנהל. אם ננסה להפוך אותו לכהן, נקבל שגיאה:

```
'var p: Principle = t; // Type 'Teacher' is not assignable to type 'Principle'
```

זאת מכיוון שהמהדר זהה ניסiou לישם ערך מסווג **Teacher** למשתנה מסווג **Principle**. עם זאת, אנו יודעים להניד שמדובר בפקודה תקינה ולכן, כדי לעקוף את המהדר, נצמיד למשתנה **t** את סוג הנתונים **Principle**, באמצעות הפקודה **as**:

```

2 var p: Principle = t as Principle; // valid!
3
4 p.startDate = new Date("2016-07-08");

```

הצמדת סוג נתונים שתאפשר רק כאשר סוג הנתונים **<t>** הוא תת-סוג של **<p>** או להפוך. לעומת זאת, הצמדת סוג נתונים שתאפשר רק מכיוון ש - **Teacher** מרחיב את **Principle**.

הצמדת סוג נתונים כפולה

אם נרצה לבצע הצמדת סוג נתונים עבור שני סוגי נתונים שונים, נבצע **הצמדת סוג נתונים כפולה**:

```

1
2 var age: string = "15";
3 var numericAge: number = age as any as number;
4

```

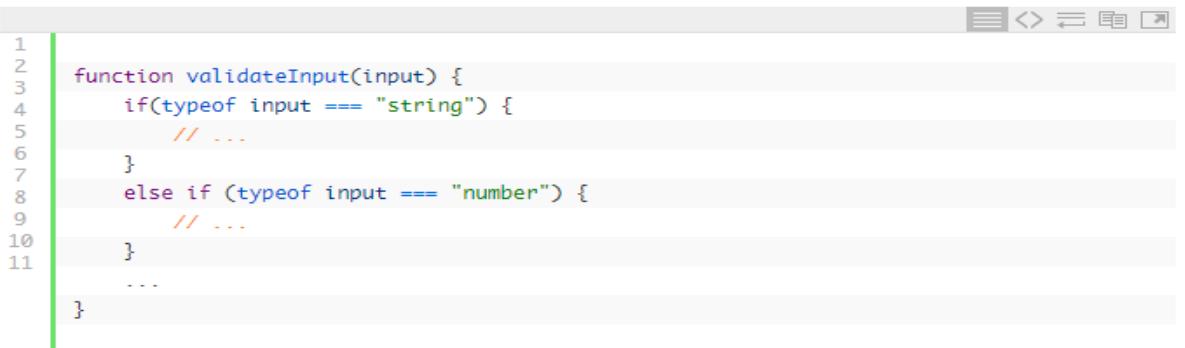
סוג הנתונים **any** יכול להציג או להיות מוצמד לכל סוג נתונים אחר ולן ניתן להשתמש בו כסוג "מעבר" בין שני סוגי ללא קשר ביניהם.

חשוב לציין שהצמדת נתונים בכל והצמדת נתונים בפרט נחשים לאמצעים "קיצוניים" ויש להשתמש בהם בחכמה, מכיוון שהם "مبטלים" את המהדר. תוכנה זאת נועדה בעיקר כדי לאפשר למתכנתים לשלב קטיעי זו.

65

אין ספק שבשפה כל כך דינמית כמו JavaScript, עקרון העמסה הוא חשוב מאוד מכאן כמוהו. מספקת טכניקת העמסה פשוטה וモכרת, שמיושמת בשפות כמו JAVA – SWIFT – TypeScript.

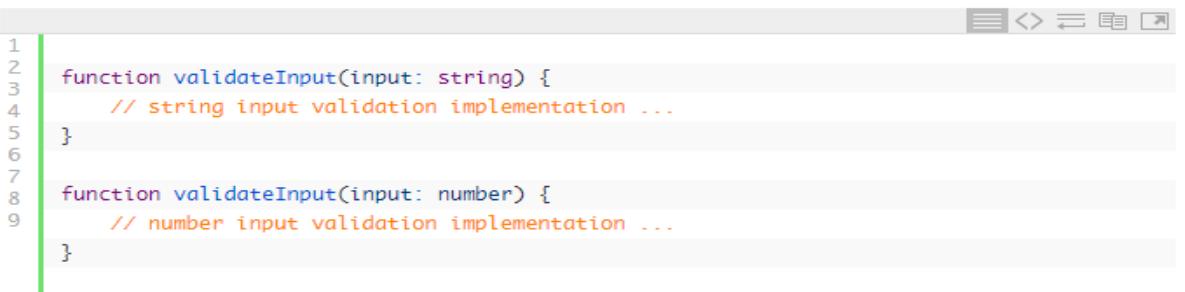
אם ב – JavaScript אנחנו נאלצים לנתח את הפרמטר בגין הפונקציה:



```

1
2     function validateInput(input) {
3         if(typeof input === "string") {
4             // ...
5         }
6         else if (typeof input === "number") {
7             // ...
8         }
9         ...
10    }
11 }
```

ב – TypeScript, העבודה תיעשה בשבילנו:



```

1
2     function validateInput(input: string) {
3         // string input validation implementation ...
4     }
5
6     function validateInput(input: number) {
7         // number input validation implementation ...
8     }
9 }
```

וכשלוקחים בחשבון את היכולת ליצור סוג-נתונים מותאמים אישית, העמסה הופכת ליכולת שימושית יותר ש –
תאפשר לנו המפתחים TypeScript לננתן לנו המפתחים.

זה בדיק העיתוי המושלם להתעמק בסוגי-נתונים מותאמים אישית, או בשם הרשמי:

מילה המפתח **this** מאפשרת לגשת לשדות ולמетодות בתוך האובייקט שבו היא נמצאת.

בדוגמה הבאה, השדה `message` מכיל את המחרוזת שתחזיר המethode `sayHello`, והmethode יכולה לגשת לשדה באמצעות `this`.

```
class Car {
    message: string;

    sayHello(){
        return this.message;
    }
}

// יצור אובייקט מהמחלקה
let myCar = new Car();
```

נגידו את המסר שאנו רוצים שהmethode תחזיר, אז נקרא לmethode.

```
myCar.message = "Howdy-doody!";
console.log(myCar.sayHello()); // Howdy-doody!
```

ואנו שמתם לב שהגדכנו את הערך של המשתנה `message` מחוץ למחלקה, זהה מפנוי שהמשתנה `message` הוא משתנה ציבורי (**public**).

במידה ונרצה למנוע גישה ישירה לשדה נצטרך לשנות את ההגדרה שלו לפרטי (**protected**) או פרטי (**private**).

```
class Car {  
    private message: string;  
  
    sayHello(){  
        return this.message;  
    }  
  
    setMessage(text: string){  
        this.message = text || "Howdy-doody!";  
    }  
  
    getMessage(){  
        return this.message;  
    }  
}  
  
let myCar = new Car();  
  
myCar.setMessage ("Beep! Beep!");  
  
console.log(myCar.sayHello()); //Beep! Beep!  
  
console.log(myCar.getMessage()); //Beep! Beep!
```

הגדכנו את ערכו של המשתנה message באמצעות `.setMessage`.
`setMessage` יכול לקבל רק ערכים מסווג מחרוזת, ובמידה ואינו מקבל ערך, יקבל ערך ברירת מחדל.

אנו יכולים למנוע את האפשרה למשתנים מחוץ למחלקה על ידי הוספת הקידומת **private**

```
class Car {  
    private message: string;  
  
    sayHello(){  
        return this.message;  
    }  
}
```

במידה ונגיד רשותה כפרט, לא יוכל לגשת אליו מחוץ למחלקה, והדרך היחידה לגשת אליו היא באמצעות פונקציות ציבוריות מסווג **getter** ו-**setter**.

- פונקציות **setter** מאפשרות לקוד מחוץ למחלקה לשנות את ערכו של משתנה פרט.
- פונקציות **getter** מאפשרות לקוד מחוץ למחלקה לקבל את ערכו של משתנה פרט.

פונקציית בניין (constructor)

פונקציית בניין היא הפונקציה הראשונה שרצה בכל פעם שיוצרים אובייקט חדש של מחלקה, וכך ניתן להשתמש בה כדי להציג ערכים חיוניים לאובייקט.

בדוגמה הבאה, המשתנים `model` ו-`year` הכרחיים לפעולות האובייקט, וכך נציג אותם כבר בזמן שאנו יוצרים את האובייקט מהמחלקה.

```
class Car {
    constructor(public model: string, public year: number) {}
}

let sussita = new Car("Carmel", 1979);
console.log(sussita.model, sussita.year); // Carmel, 1979
```

שימוש לב, שמספיק להגיד את המשתנים `year` ו-`model` רק פעם אחת כפרמטרים בפונקציית הבנייה כדי שהם יפכו לשודות לכל דבר באובייקט (`sussita`) שייצרנו מהמחלקה. ומכיון שהמשתנים מוגדרים כמשתנים ציבוריים (`public`), ניתן לגשת אליהם ולקבל את ערכם (השורה الأخيرة בדוגמה).

שדות ומетодות סטטיים

```
class Car {
    static numberOfCars: number = 0;

    constructor(public model="", public year=0){
        Car.addOne();
    }

    static addOne(){
        Car.numberOfCars++;
    }
}

let sussita = new Car("Carmel", 1979);
let subaru = new Car("Subaru DL", 1989);

console.log(Car.numberOfCars); //2
```

שדות ומетодות סטטיים משמשים לביצוע פעולות ולמעקב אחר תוכנות שאינן שייכים לאובייקט מסוים אלא לכל האובייקטים שנוצרו מהמחלקה. לכן, משתמש בשדות ובMETHODS STATICיים לשימושים כלליים (utility).

לדוגמה, אם מעוניין לדעת כמה אובייקטים שיצרנו מהמחלקה. נגדיר משתנה סטטי `numberOfCars`, ונitin לו ערך ראשוני של אפס, ובכל פעם שנוצר אובייקט חדש מהמחלקה נגדיר שmethod הבנייה ווסף 1 למשתנה זה.

ניתן ליצור מחלקה שירשת ממחלקה אחרת את כל התכונות והmethodות. במצב זה, המחלקה הירשת:

- מקבלת את כל התכונות והmethodות של המחלקה שאוותה היא ירשת.
- יכולת להוסיף תכונות וmethodות.
- יכולת לדרס תכונות וmethodות שהיא ירשה.

נשתמש במילה השמורה **extends** כדי לציין ירשה.

בדוגמא הקוד להלן, המחלקה SportsCar ירשת את המחלקה Car, וגם המחלקה היורשת דורשת את הקונסטרוקטור של המחלקה המקורית.

```
class SportsCar extends Car {
    // The constructor overrides the parent constructor
    constructor(public model:"", public year=0){
        super(model, year); // Derived constructors must
        contain a super call
        SportsCar.addOne();
    }
}
```

יצור אובייקט מהמחלקה:

```
let Ferrari = new SportsCar("Ferrari", 2017);
console.log(Car.numberOfCars); // undefined
// אין בעיה להשתמש במתודה Car.numberOfCars
```

האם האובייקט שיר למחלקה Car?

```
console.log(Ferrari instanceof Car); // true
```

תכונות המחלקה שיזכו למקדם **protected** יהיו נגישות אך ורק מתוך המחלקה ומתוך מחלקה מרחיבת, אך לא מוחוץ למחלקה:

```

1
2 class Input {
3   protected type: string;
4   protected dirty: boolean;
5
6   constructor(type: string, dirty: boolean) {
7     this.type = type;
8     this.dirty = dirty;
9   }
10 }
11
12
13
14
15 class ageInput extends Input {
16   private value: number;
17   private ageLimit: number;
18
19   constructor(age: number, dirty: boolean, ageLimit: number) {
20     super("age", dirty);
21     this.value = age;
22     this.ageLimit = ageLimit;
23   }
24
25   public validate(): boolean {
26     return this.dirty && this.value >= this.ageLimit;
27   }
28 }
29
30
31
32 let ageInput1 = new ageInput(17, true, 21);
33 ageInput1.validate(); // false
34 ageInput1.type; // error!

```

بعد שהפונקציה **validate** תפעל כצפוי, מכיוון שהיא נגישה למשתנים המוגנים מתוך המחלקה, הניסון לנשחת למשתנה **type** מוחוץ למחלקה ויזיק מיד את המהדר, שיתריע בפנינו מיד: אין לכם גישה למשתנה זה מוחוץ למחלקה!

Property 'type' is protected and only accessible within class 'Input' and its subclasses

עד כה, אין חדש תחת השם. והרי לא באתי לשעכם אתכם, ורק אגלה לכם על היכולת הסודית (לא באמת, אבל הרבה אנשים לא מודעים אליה) ש - protected קיבל מפתחו TypeScript: למן עיצור מופע של מחלוקת מחוץ למחלוקת המרחביה.

ניקח את הדוגמה מלמעלה, והפעם נוסיף **protected** גם לבנאי:

```
1 class Input {
2     protected type: string;
3     protected dirty: boolean;
4
5     protected constructor(type: string, dirty: boolean) {
6         this.type = type;
7         this.dirty = dirty;
8     }
9 }
10
11
12
13
14
15 class ageInput extends Input {
16     private value: number;
17     private ageLimit: number;
18
19     constructor(age: number, dirty: boolean, ageLimit: number) {
20         super("age", dirty);
21         this.value = age;
22         this.ageLimit = ageLimit;
23     }
24 }
25
26
27 let input1 = new Input("text", false); // error
```

השניה שמתבקשת תודיע לנו שפונקציית הבנאי לא גישה מחוץ למחלוקת:

Constructor of class 'Input' is protected and only accessible within the class declaration

למרות זאת, המחלוקת **can** ניתנת להרחביה.

כאשר נרצה להגדיר תכונה שיכולה לקבל ערך רק פעם אחת, במעמד יצירת המופיע, נעשה זאת באמצעות המילה השמורה `:readonly`

```
1 class Input {  
2     <strong>readonly</strong> type: string;  
3     protected dirty: boolean;  
4  
5     protected constructor(type: string, dirty: boolean) {  
6         this.type = type;  
7         this.dirty = dirty;  
8     }  
9 }  
10  
11 }
```

. TypeScript מאפשרת להזכיר על פרמטר `C` – ובכך להפוך אותו לתוכנה.
אםنم האפשרות ההז חוסכת שורת קוד, אבל כנראה שבכל הנוגע לסדר בקוד אני קצת שמן ולא מתי על האפשרות זו, אבל כדי שלא תגידו שאני מסתור מידע, קבלו דוגמה:

```
1 class Input {  
2     protected dirty: boolean;  
3  
4     protected constructor(readonly type: string, dirty: boolean) {  
5         this.dirty = dirty;  
6     }  
7 }  
8  
9 }
```

interface בסגנון מונחה עצמים ב-TypeScript

74

מחבר: יוסי בן הרוש בתאריך: 24.02.2017

מחייב כל קוד שמיישם אותו לכתוב מותודות או שדות מסוימים.

זה יכול להוועיל כשרוצים לחייב את כל המתכנתים בצוות לכתוב קוד בסגנון אחד, או שכותבים קוד ואות"כ רציתם להזכיר מה המבנה שלו.

אפשר להמשיך לנסוטה להסביר את זה, אבל אני מעדיף להציגם באמצעות קוד.

```
interface Car {  
    model: string;  
    year: number;  
    color?: string; // שדה אופציוני  
}
```

עכשו כל אובייקט או מחלקת שתשים את ה-interface **Car** ישמו Car יהיו חיברים להכיל את השדות `model` ו-`year`, אבל לא את השדה `color`, כיון שהוא אופציוני (כפי שמעיד סימן השאלה שモופיע מימין לשדה).

נאמר שה משתנה `subaru` מישם את ה-interface

```
let subaru: Car;  
  
subaru = {model: 'DL', year: 1989};
```

קוד זה יעבור קומpileציה בהצלחה מפני שהוא מכיל את שני השדותשה-interface מחייב אותם להציג.

```
subaru= {model: 'DL', year: 1989, color: 'beige'};
```

גם קוד זה יקומפל בהצלחה כי הוא מכיל את שני שדות החובה וגם את הפרמטר האופציוני.

```
subaru= {model: 'DL', year: 1989, color: 'beige',  
driver: 'Moshe'};
```

הקומPILEציה של הקוד תזורך שגיאה, בגלל שהוספנו שדה שלא קיים ב-interface `(driver)`.

```
subaru = {model: 'DL'};
```

קוד זה יגרום לשגיאת קומPILEציה כי השדה `year` החינו ליישום ה-interface חסר. כפי ש-interface יכול לחייב אובייקט להכיל שדות מסוימים, הוא יכול לחייב מחלוקת ליישם שדות ומותודות מסוימים.

```
// Interface
interface Car {
    speed: number;
    accelerate(by: number): void;
}

// Class to implement the interface
let car: Car = {
    speed: 80,
    accelerate: function(by: number): void{
        this.speed += by;
    }
}
```

מה יקרה כשנקملפּ את הקוד ל-JavaScript

קומpileציה של הקוד ל-JavaScript תגרום להעלמתה ה-interface, וזה בכלל שאין TypeScript שום דבר שמקביל ל-interface שמציע JavaScript.

כך יראה הקוד המקורי:

```
האינטרפייס נעלם בקומPILEציה //
var car = {
    speed: 80,
    accelerate: function (by) {
        this.speed += by;
    }
};
```

אם כן מה הטעם להשתמש ב-interface?

כדי להשתמש ב-interface כיוון שהוא מוסיף את ארגון קוד ה-TypeScript, והוא שימושו במיוחד כשעובדים בצוותים גדולים ורוצים לחיבב את כל מי משתמש בקוד לכתוב מתודות מסוימות. בנוסף, ב-[Angular 2](#) יש שימוש ב-interface, ולכן חשוב להכיר את הטכניקה.

במדריך הבא, נטפל בעוד טכניקה מועילה של TypeScript שהוא השימוש ב-generics.

76

```

1
2 interface Message {
3   title: string;
4   content: string;
5   to: string;
6
7 }
```

נשתמש בהצמדת נתונים בשביל לשיר את הממשק:

```

1
2 function sendMessage(messageInstance: Message) {
3   // ...
4
5 }
```

כעת, אם ננסה לספק לפונקציה כפרמטר אובייקט שלא תואם את דרישות הממשק:

```

1
2 let messageInstance = sendMessage({title: "Hello Master", content: "Any pizza leftover?"});
3
4 }
```

נקבל שנייהה:

Argument of type '{ title: string; content: string; attachments: {}; }' is not assignable to parameter of type 'Message'

בדוגמה ניתן לראות שنم הוסיף משתנים מעבר למה שהוגדר במשק לא אפשרית, וCOMMON שוחזרת משתנים גם היא פשע שעליו הקומpileר לא סולח!

אבל איך בכלל זאת ניתן עם תכונות כמו מכותבים, שלעתים יעשה בהן שימוש ולפעמים לא? יש לנו תחושה שאתם כבר יודעים את התשובה..

טייאור תכונה במשמעות כתוכה אופציונלית תיעשה באמצעות הוספת סימן שאלת (?) בסוף השם של התכונה:



```

1
2 interface Message {
3   title: string;
4   content: string;
5   to: string;
6   <u>cc?:</u> string;
7 }
8

```

כעת, המהדר יקבל את הדוגמה לעיל, עם התכונה **cc**, אבל אפשר לנו גם לוויתר עליה.

תכונות לקריאה בלבד

אנו מכירים את הלוויי **readonly** מהמחלקות, ונעם במשקיים הוא לא נוטש אותנו:



```

1
2 interface UserStructure {
3   readonly id: number;
4   username: string;
5   email: string;
6   password: string;
7 };
8

```

ולמי שכח, תזכורת: **readonly** מנדר תכונה שיכולה לקבל ערך רק פעם אחת, במעמד יצירת המופיע.

כמו בכל שפת תכנות שמשקדים קיימים בה, גם TypeScript מאפשר לתאר פונקציות במשך. בשילוב עם זאת, נספיק לרשום את שם וחתימת הפונקציה, שמורכבת מהפרמטרים שהוא מקבלת וטיפוס הנתונים שהיא מחזירה:

```

1
2 interface UserStructure {
3   readonly id: number;
4   username: string;
5   email: string;
6   password: string;
7
8   editEmail(email: string): void
9 }
10

```

ניתן גם להשתמש במשך כדי לתאר פונקציה (ולא מחלוקת). במקרה כהו נותר על שם הפונקציה, ונספיק לרשום את החתימה:

```

1
2 interface EditUserEmail {
3   (email: string): void
4 }
5

```

"שימוש הפונקציה ייראה כך:

```

1
2 let editEmail: EditUserEmail = function(email: string) {
3   // ...
4 }
5

```

נכיר על יישום ממשק במחלקה באמצעות המילה השמורה **:implements**:

```

1 class User implements UserStructure {
2   readonly id: number;
3   username: string;
4   email: string;
5   password: string;
6
7   editEmail(email: string): void {
8     // ...
9   }
10 }

```

כמובן שם לא נישם את המשק במלואו, מיד המהדר יכנס לפעולה וידאג לעדכן אותו על כך:

Class 'User' incorrectly implements interface 'UserStructure'

הרחבת ממשק

ניתן להרחיב ממשק בדומה לכך שהוא מרחיבים מחלקה – באמצעות המילה השמורה **:extends**

```

1 interface UserStructure {
2   readonly id: number;
3   username: string;
4   email: string;
5   password: string;
6 };
7
8 interface AdminStructure extends UserStructure {
9   superadmin: boolean
10 }

```

כפי שניתן לראות בדוגמה, אין צורך לחזור על התכונות (או הfonקציות) של המשק המורחב במשק המרחיב – הוא יורש אותן.

משק מרחיב מחלקה

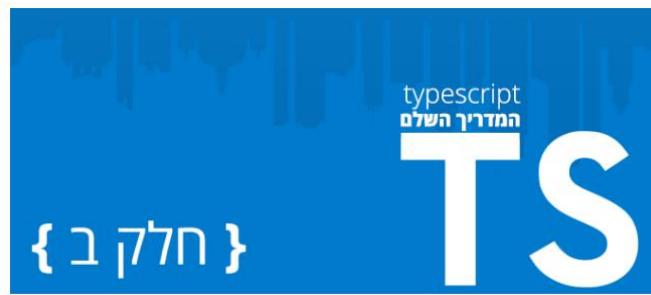
TypeScript מאפשר למשק להרחיב מחלקה ובעצם לרשת את תכונותיה והfonקציות שלה, גם כן באמצעות המילה השמורה **:extends**

```

1 class User {
2   readonly id: number;
3   username: string;
4   email: string;
5   password: string;
6
7   edit(property, value): void;
8   remove(): void;
9 }
10
11 interface UserStructure extends User {
12   // ...
13 }

```

המשק יורש את כל התכונות והfonקציות של המחלקה, **לא היישום**.
יכולת זו עשויה להתגנות כמשמעותית בעבודה עם ספריות צד-שלישי.



עד קצת על מחלקות ב - ..TypeScript

פונקציות עדכון / גישה לתוכנה

(באנגלית זה נשמע יותר טוב: **Mutator / Accessor Method**)

הכוונה היא לפונקציות שונות לנו שליטה על תהליכי עדכון / גישה ל ערךן של תוכנות מחלקה, ובוציאות אשר אנו מנוטים לגשת / עדכן משתנה מחלקה מחוץ לה:

```
1 let myDate = new DateFormat("d/m/Y"); // set method executed
2 console.log(myDate.dateEntry); // get method executed
```

השימוש של הפונקציות הללו נעשה באמצעות שתי מילויים שモורות: **get** לשילטה על גישה לערך תכונה – **set** לשילטה על עדכון ערך תכונה.

בדוגמה הבאה, שמדינימה איך לישם פונקציות מסוג זה, מוגדרת מחלוקת `DateFormat` שומרה גם את

כיה:

```
1
2
3
4 class DateFormat {
5   private _dateEntry: Date;
6   private format: string;
7
8
9   constructor(format: string) {
10     this.format = format;
11   }
12
13
14   get dateEntry(): Date {
15     return this._dateEntry;
16   }
17
18
19   set dateEntry(value: Date) {
20     let formatStr = this.format.replace(/\//g, ''); // strip slashes
21     this._dateEntry = this.doFormat(value, formatStr);
22   }
23
24
25
26   private doFormat(value: Date, format: string): Date {
27     return this[`format_${format}`](value);
28   }
29
30
31
32   private format_dmY(value: Date): Date {
33     // implementation: format date to "d/m/Y"
34   }
35
36
37   private format_mdY(value: Date): Date {
38     // implementation: format date to "m/d/Y"
39   }
40
41
42   // ...
43 }
44
45
46
47 let myDate = new DateFormat("d/m/Y");
48 console.log(myDate.dateEntry);
```

```

1
2 class DateFormat {
3     private _dateEntry: Date;
4     private format: string;
5
6
7     constructor(format: string) {
8         this.format = format;
9     }
10
11 // ...
12 }
```

בחלק זהה, הוגדרו שתי תכונות מחלקה:

_dateEntry •
format •

שםו לב שתי התכונות ה- **private** ולכן לא ניתן לנשחן אליהן מחוץ למחלקה.
בנוסף, הנדרנו פונקציית בניית שמקבלת כפרמטר מחזרות עם הפורמט הרצוי של התאריך.

—
אליה מכם שעוקבים אחרי הפוסטים שלי בטח מודעים – איך מישחו שכטב את הפוסט **PHP** – לכתוב את זה **נכון** מרצה לעצמו להזכיר על משתנה **private** אחד עם קוו תחתון, ומשתנה **private** שני בלי קו תחתון?!
התשובה – ממש בהמשך 😊

```

1
2     get dateEntry(): Date {
3         return this._dateEntry;
4     }
5
6
7     set dateEntry(value: Date) {
8         let formatStr = this.format.replace(/^\//g, ''); // strip slashes
9         this._dateEntry = this.doFormat(value, formatStr);
10    }

```

בקטע הקוד לעיל, קוראה הקסם:

הנדכנו שתי פונקציות עם מקדם `.set / .get`

הישום פחות קריטי, אך חשוב לדעת ולהבין שלושה כללים של המהדר, בהקשר של פונקציות `get / set`:

1. פונקציית `get` לא יכולה לקבל פרמטרים

2. פונקציית `set` יכולה לקבל פרמטר אחד בלבד

3. סוג הערך **משמעותו** פונקציית `get` – `set` חייב להיות לסוג הפרמטר שפונקציית `get` – **מקבלת**

במידה וננדיר עבור משתנה פונקציית `get` בלבד, המהדר יתייחס אליו אוטומטית כ – `readonly` – קלומר –
משתנה שיכל לקבל ערך רק פעם אחת, במעמד יצירת המופיע.

אני יודע, אני יודע.. אתם כבר לא עומדים במתוח.

משתנה `private` אחד עם קוו תחתון ומשתנה `private` שני בלי קו תחתון?! הסיבה לכך פשוטה:

המנגינה כולם **לא להשתמש** בקו תחתון מקדם ל – `private`, וניתן לראות שגם יוצר TypeScript
מקפידים על כך בדוקומנטציה.

עם זאת, מכיוון ש `get` ו `set` מוגדרים פרטנית למשתנה (בניגוד ל – PHP למשל),

אם לא "נסמן" את המשתנה, יוצר מצב שבו יש לנו תוכנה ופונקציה זהה. ועל זה, המהדר לא ישtopic.

מאפשרת הכרה על תכונות סטטיות, באמצעות המילה השמורה `static`.

כל תכונה יכולה להיות בנוסף גם `readonly`, `public` / `protected` / `private` וכו'!

```

1
2 class Grid {
3     private static origin = {x: 0, y: 0};
4     calculateDistanceFromOrigin(point: {x: number; y: number;}) {
5         let xDist = (point.x - Grid.origin.x);
6         let yDist = (point.y - Grid.origin.y);
7         return Math.sqrt(xDist * xDist + yDist * yDist) / this.scale;
8     }
9     constructor (public scale: number) { }
10 }
11
12 let grid1 = new Grid(1.0); // 1x scale
13 let grid2 = new Grid(5.0); // 5x scale
14
15 console.log(grid1.calculateDistanceFromOrigin({x: 10, y: 10}));
16 console.log(grid2.calculateDistanceFromOrigin({x: 10, y: 10}));
```

(הדוגמה נלקחה מהdockumentation של TypeScript ונורכה)

בקטע הקוד מוכח משתנה סטטי `origin`, והגישה אליו נעשית באמצעות שם המחלקה ולא `this`.

אם ננסה להשתמש ב- `this`, נקבל את השגיאה הבאה:

'Property 'origin' does not exist on type 'Grid'

נית להגדיר גם פונקציות סטטיות, באותו האופן שבו אנו מגדירים תכונה סטטית:

```

1
2 static myFunc() {
3     // ...
4 }
5 }
```

מחלקות מופשטות

ב - TypeScript, מחלקה מופטשת (Abstract Class) פועלת לפי אותו עקרון כמו במרבית שפות התכנות: היא מחלקה שלא ניתן ליצור מופעים מסוגה והוא מיועדת להיות בסיס למחלקות אחרות, כשההבדל בין מחלקה מופטשת למשק טמון בכך שבמחלקה מופטשת ניתן למשת פונקציות, בניגוד למשק.

נכיר על מחלקה או על פונקציה כמופטשת באמצעות המילה השמורה `:abstract`

```

1 abstract class Greeter {
2   protected _greetMessage: string;
3
4   constructor(greetMsg: string) {
5     this._greetMessage = greetMsg;
6   }
7
8   public greet() {
9     alert(this.greetMessage);
10  }
11
12  abstract get greetMessage(): string;
13  abstract set greetMessage(message: string);
14}
15
16

```

אם נעד לאתגר את המהדר ע"י ניסין ליצור מופע מסווג Greeter, מיד הוא ינדוף בנו:

'Cannot create an instance of the abstract class 'Greeter'

שימוש במחלקה ממשק

מכיוון שאפשר לנו מגדירים מחלקה, אנחנו גם מגדירים סוג נתונים, TypeScript מאפשרת להתייחס למחלקה ממשק, בין אם כבסיס למשק אחר או כישום במחלקה:

```

1 class A {
2   // ...
3 }
4
5 class B implements A {
6 }
7
8 interface C extends A {
9 }
10
11
12
13

```

כל תוכנת שמכבד את עצמו שואף לכתוב קוד גנרי שניין יהיה לעשות בו שימוש חוזר. בשפה כמו Javascript החשיבות של ארנו-כלים באמצעותו נוכל להתמודד עם סוג נתונים לא ידוע היא עצמה, על אחת כמה וכמה בשפה שמתוירת (מצליהה) ללקת-ID-ביד עם ספירות Javascript שכותבות בנייסיב. הנדרת סוג נתונים גנרי יכולה להתבצע בעבר: פונקציות, ממשקים, מחלקות ותוכנות מחלוקת.

פונקציה גנרטית

```
1 function identity<V> (arg: V): V {
2   return arg;
3 }
4
5 }
```

התחביר דומה מאד ל - C++, אבל גם מוכנת C# ו - JAVA ימצאו בו משהו מוכר. הפונקציה מוגדרת כגנרטית ע"י הצמדת `<T>` לשמה: המשמעות בפועל היא שללא ההצמדה, המהדר לא יכיר בסוג נתונים `T` והפונקציה לא תוכל להיות גנרטית.

ממשק גנרי

כל הנוגע לממשקים (וגם למחלקות), ניתן להגדיר את הפורמטור הגנרי בחתימת הממשק:

```
1 interface GenericOne<T> {
2   property: T; // generic property
3   (arg: T): T; // Generic method
4
5 }
6 }
```

וכך הסוג יהיה נגיש עבור כל חברי הממשק.

לעומת זאת, נוכל לבחור להגדיר את הפורמטו הגנרי ספציפית עבור פונקציה:

```

1
2 interface GenericFn {
3   &lt;T&gt;(arg: T): T; // Generic method
4 }
5 
```

מחלקה גנריית

העקרון זהה לממשקים, הנדרת הפורמטו תישא או בחתימת המחלקה:

```

1
2 class GenericTwo<E> {
3   private gProperty: E;
4
5   public gFunc(x: E): E {
6     return this.gProperty;
7   }
8 }
9 
```

או בפונקציה עצמה:

```

1
2 class GenericTwo {
3   public gFunc<E>(x: E): E {
4     return x;
5   }
6 }
7 
```

נכון, אבל לסוג הנתונים `any` יש חסרון ממשמעותי, כאשרחנו מדברים על גנריות: **הוא לא ננרי**. הוא יכול

להכיל כל סוג נתוניים.

גם אם הוא מכיל מספר או בוליאני או סוג נתוניים שיצרנו בעצמנו – אין דרך לדעת!
זהות!

המהדר יתיחס אליו בדיקן כמו שהוא – כל סוג נתוניים אפשרי – מה שהופך את השימוש בו ללא בטוחה.
כדי להמחיש למה, נסתכל על קטע הקוד הבא, בו מתחזר פונקציה שבודקת את אורך סוג הנתוניים,
ומקבלת כל סוג נתוניים:

```
1 function getLength(arg: any): number {
2   console.log(arg.length)
3   return arg;
4 }
5
6
7
8 getLength({}); // runtime error
```

מכיוון שהמהדר מתעלם מסוג הנתונים `any`, הוא לא יזהיר אותנו מהאפשרות שלפרמטר `arg` אין תכונה
בשם `length`.

והשגיאה תתגלה רק בזמן ריצה. זה לא טוב או רע, זו ההתנהגות המצופה מסוג נתוניים `any`.

—
T ... E ... T ... E ... מה קורה פה?

שם הפרמטר הננרי (זה שנמצא בין החצים `<>`) ניתן לקביעה ע"י המתכנת ואין לו ממשמעות אמיתית.
או למה אתם נתקלים באוון אותן שאלות? מדובר במסכמה ולא בחובה, קובלן את המשמעות
שליהם:

(Type : T)

(Event : E)

(Value : V)

סוג נתונים ננרי מרחיב

כאשר נרצה לחיב את סוג הנתונים הננרי לעמוד בדרישות ספ', נוכל להגדיר אותו כמרחיב ממשק

באמצעות המילה השמורה `: extends`

```
1 interface Example {
2   // ...
3 }
4
5 class A<T> extends Example<T> {
6   // ...
7 }
```

כעת, סוג הנתונים `T` חייב לכךים את הממשק `Example`.

סוג נתונים מוצלב

(באנגלית: Intersection Type)

ב - Javascript, לעיתים קרובות נכתב קטע קוד שמסוגל יכולות של שני אובייקטים ומאנך אותם לאובייקט אחד.

ב - TypeScript מספק לנו מנגןון שמאפשר לנו לעשות זאת באופן בטוח.

באפשרות, וודקה לשוטפות חמוץ של אשלומקה על גבשו לזרוםבה. כן כן, אותן תהיליך שלמדנו עליי ביסודי (וهمשוגעים לדבר גם בתיכון...) באוניברסיטה?!) – מהליך רבית בשביל להבין יותר טוב את כוונת המשורר, ניזכר בתהיליך האבקה.

אם יש ביולוגים בקהל, אני מתנצל מראש על השגיאות הלוגיות, הפטאליות, כתיב וכל שאר השגיאות שאעסה בתיאור תחילת ההאבקה!

בהתילר ההאבקה משותף הצמח (Plant, בקטע הקוד) וחרק, במקורה שלנו – דבורת דבש (HoneyBee). הצמח, מחד, מייצר צוף – שמכיל בתוכו את האבקנים המשמשים לרבייה הדבורה, מאידך, יונקת את הצוף ומעבירה אותו לתא הרבייה של הצמח.

ועכשיו, מעברית לך:

```
function extend<T, U>(first: T, second: U): T & U {
    let result = {};
    for (let id in first) {
        result[id] = first[id];
    }
    for (let id in second) {
        if (!result.hasOwnProperty(id)) {
            result[id] = second[id];
        }
    }
    return result;
}

class Plant {
    private nectarLevel: number;

    constructor() {
        this.nectarLevel = 0;
    }

    public producePollen(amount: number): void {
        if(amount > 0)
            this.nectarLevel += amount;
    }
}

class HoneyBee {
    public suckAndPut(nectar: number): void {
        // ...
    }
}

let pollination = extend(new Plant(), new HoneyBee());
```

הפונקציה הינה יישום של תבנית מוכרת, ותפקידה ליצור סוג נתונים מוצלב. החלק המשמעותי יותר בפונקציה, בהתייחס למדריך זהה, הוא החתימה שלה:

```

1 function extend<T, U>(first: T, second: U): T & U
2
3

```

- הפונקציה **מקבלת** כפרמטרים שני סוגי גנריים, **ע' , T**
- הפונקציה **מחזירה** עצם מסוג **T & U** – עצם מסוג נתונים מוצלב.

איחוי סוגי הנתונים נעשה באמצעות האופרטור **&**

HoneyBee - | Plant

שתי מחלקות שאינן בינהן קשור יושה לניטמי, אך יכולות של שתיהן נדרשות לצורך ביצוע תהליכי מסויים – בדיקת המטרה לשמה ועוד סוג נתונים מוצלב.
אםنم לא כתבתי יישום מלא של המחלקות, אבל הנה הסבר על ה"יכולות" של כל אחת מהן, בנדול:

pollination

בסוף קטע הקוד,
mongdr משתנה **pollination** שמכיל את סוג הנתונים המוצלב, באמצעות קריאה לפונקציה **extend**
והעברת מופעים של כל אחת מהמחלקות כפרמטרים.
המופיע מכל את כל התכונות והפונקציות משתי המחלקות.

(באנגלית: Union Types)

TypeScript מziaהה להתמודד עם מצבים בהם סוג הנתונים לא ידוע, אך כן ידוע אילו סוגי נתונים הוא יכול להיות.

ничustum נכו... זה נקרא סוג נתונים משולב, והוא אפשר למהדר מצד אחד לתאום את המזיאות (כמו שאמרתי לא פעם, TypeScript עובדת טוב עם ספריות Vanilla) אך מצד שני לא לוותר על עצמו, כמו שקרה עם סוג המשתנים any.

התחבר להגדרת סוג נתונים משולב פשוט מאד: ... | type1 | type2 | type3 | ...

רוצים דוגמה? קיבלתם

בקטע הקיים הבא מתוארת מחלוקת שMRI שורות פקודה (CLI) – הממשק אפשר הכנסת פקודה בכל פעם, או מערך של פקודות.

```
1  interface CLI {
2    program: string;
3
4    execute(command: string | string[]): boolean | string;
5  }
6
7  class CommandLine implements CLI {
8    program: string;
9
10   public execute(command: string | string[]): boolean | string {
11     // command execution implementation
12   }
13 }
14
15 }
```

כעת, כאשר נעביר כפרמטר מחזרות או מערך של מחזרות, הכל יעבד כמצופה. אם נעביר כפרמטר מספר או כל סוג נתונים שהוא לא מחזרות / מערך מחזרות?.. אתם כבר יודעים שהקומפיילר לא אוהב דברים אלה:

Argument of type 'number[]' is not assignable to parameter of type 'string | string[]'

זה עוד לא הכל! המהדר ידע להציג לנו רק פונקציות/תוכנות שזמיןות עבור סוגי הנתונים הרלוונטיים ואם השתמש בפונקציה שלא זמינה – קיבל מיד הודעה שנייה.

נקודה אחורונה: חד-העין מביניכם בוודאי הבחינו שגם בסוג הנתונים המוחזר, בחתימת הפונקציה, נעשה שימוש בסוג נתונים משולב. ואם לא שמתם לב – רשמו לכם שגם סוג נתונים משולב ניתן לשימוש גם בהחזרת נתונים.

סמלים

(באנגלית: **Symbols**)

לפנינו שאבסיר על סוג הנתונים עצמו, חשוב לציין שמדובר בסוג נתונים חדש, שהווסף לתקן

ECMAScript 2015

והמהדר ידע לעבוד איתו ורק אם נדרש לו שאנחנו עובדים לפי התקן **es6**:

```
1 | "target": "es6"
```

בכינוד למחלקות, ממשקים וכו' שמומרים לקוד דן שנותר באופן רחב, סוג הנתונים החדש לא נתרם בכלל ב – EO על כל גרסאותיו, וחלק מתוכנותיו לא נתמכות נס באופרה ובසפארי. (המידע נלקח מ – [mozilla.org](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Symbols).**טלבלת תאמות לדפדפניים של סוג הנתונים**). בנוספף, הישום של סמלים ב – TypeScript הינו חלק, ככל הנראה בשbill למונע אי-תאמות כל הניתן.

אני, באופן אישי, משתמש בשלב זה לא להשתמש בסוג הנתונים זהה מכיוון שאיננו נתמך באופן רצוף באחוון גבורה מספיק של דפדפניים, אבל מצד שני אני חשב ששוג הנתונים זהה טמון בחובו המון פוטנציאלי לעתיד ולכך כדי להבין אותו. אז קידמה:

סוג הנתונים הינו **פרימיטיבי** (כמו מחרוזת, מספר, בוליאני), **בלתי ניתן לשינוי** מרגע הגדרתו (בדומה לקביעים) **ובעל ערך ייחודי**.

למרות ששוג הנתונים פרימיטיבי, נדרשת הפעלת פונקציית בניית כדי ליצור מופיע שלו:

```
1 | let sym = Symbol("description");
```

הכוונה בבלתי ניתן לשינוי בחרואה, אבל למה התכוון המשורר שככתב "בעל ערך ייחודי"? בשbill להבין לעומק את ההנדשה זו, נסתכל על קטע הקוד הבא, שמתאר השוואת מחרוזות אל מול השוואת סמלים:

```
1 | let str1 = "hello";
2 | let str2 = "hello";
3 |
4 |
5 | console.log(str1 === str2); // true
6 |
7 |
8 | let sym1 = Symbol("hello");
9 | let sym2 = Symbol("hello");
10 |
11 | console.log(sym1 === sym2) // false
```

אם תבחרו להאמין לקטע הקוד לעיל, אז כמובן למסקנה: כל סמל הוא ייחודי. (ביקר לי – לא עובד עלייכם!). שתי נקודות בקשרו לשוג הנתונים:

- ב – **ECMAScript 2015** ניתן לבצע השוואת בין שני סמלים, אבל זה לא קיים ב – TypeScript נכון נוכחית.
- הערך שਮועבר כפורמט לבנייה הוא בגדר רשות, ומשמש כתיאור פנימי של הסמל, שיופיע בעת המرة למחרוזת, שימוש ב – `()` ובהודעות שנייה

קודם כל, יופי של שאלה! (קצת חנופה לא הרנה אף אחד..)

כאשר אני התיישבתי ללמידה את השפה, גיליתי שהסביר הדל במדריך הרשמי לא מאפשר להבין איך ניתן להביא לידי ביטויו את סוג הנתונים בעולם הנ'אוּוְ-סקרייפט האמתי. למזלכם – יצאתי לחקיר ברוחבי האינטרנט, וזרת עם תשובה:

"סוג הנתונים השביעי" אפשר לנו להימנע משימוש 'לא בטוח' במקרים מסוימים. ניקח לדוגמה את הסкриיפט הבא, שמסמן אלמנט DOM שבמצעת עליו מניפולציה:

```
1 function manipulator(domElement) {
2   domElement.manipulated = true;
3   // .. manipulation
4   domElement.manipulated = false;
5 }
6 
```

הכל טוב ויפה, רק שלפונקציה נקודת תורפה שימושית: אם מזוזהו סיבת (עדכן תקין), ספריות אחרות שמשתמשות באותו מזהה, ...), האלמנט כבר יגיע עם תוכנה בשם `manipulated`, היא עלולה לפנו ע בתפקיד של האלמנט והתמשקות עם ספריות אחרות. וכך באים לידי ביטוי הסמלים. זוכרים מה אמרנו עליהם? הנה תזכורת:

1. סמל הינו בלתי ניתן לשינוי מרגע יצירתו
2. סמל הינו ייחודי – לעולם לא יהיה שווה לסמל אחר

בואו נערוך את הדוגמה, ורק שהפעם נשתמש בסמל במקום במקרים מסוימים:

```
1 var manipulated = Symbol("manipulated");
2 domElement[manipulated] = true;
3 // .. manipulation
4 domElement[manipulated] = false;
5 
```

ובכן הפכנו את הסкриיפט שלנו לחסין מהתנשויות, יהיו אשר יהיו.

מעבר לצירת סמלים, TypeScript מנעה עם סט סמלים מוגדרים מראש, רשיימה שלהם [ניתן לקרוא במדריך הרשמי](#), על חלגם הקטע אסביר מיד.

ולא ת...for..of

פועלת בדומה ל`for..in` המוכרת, עם שינוי אחד קטן אך משמעותי. אתם הולכים לאהוב את זה:

כידוע, לולאת `for..in` עוברת על **אינדקסים** ולא על **איברים**:

```

1 | let myArray: Array<number> = [100, 200, 300];
2 | for(item in myArray)
3 |   console.log(item); // 0, 1, 2

```

lolala ההזו אחריות לטעות נפוצה בקרב מפתחי JS, שמצוים לקבל את הערך, ובמקום זאת מקבלים את האינדקס.

יכול להיות שהוא רק אני, אבל אני מרגש שככל שנחננו מתקדמים במעלה המדריך, כך אנחנו מתקרבים אחד לשני, ושאנחנו כבר מספיק קרובים בשביל להזוזות: כולם עשוינו את הטיעות הזאת. חלכנו איפואו חוטאים בה מיד פעם בימי עבודה בלתי נגמרם (אני? מה פתאום.. בכלל דיברתי על misuse אחר)

וכאן באהה לעזרנו לולאת `for..of`, שעוברת על **איברים**:

```

1 | let myArray: Array<number> = [100, 200, 300];
2 | for(item of myArray)
3 |   console.log(item); // 100, 200, 300

```

לא רע, נכון? 😊

הטוב

ב - **es6**, לולאת `for..of` יכולה לזרץ לא רק על מערכים, אלא על כל עצם שמודדר כנין לחזיר. בנוסף, אנחנו יכולים להנדר בעצממנו סוג נתונים שהוא ניתן לחזיר. אבל זה כברcosa שלם שקשר יותר ל - **es6** מאשר ל - **TypeScript**, ואכתוב עליו בעתיד, כשהשימוש בתוכנות של **es6** יהיה רוח יותר.

es6 עדין לא נתמך בצורה רוחבה מספיק ועבור אנשים רבים (בינם אני), זו הייתה הסיבה מלכתחילה לעבור ל – TypeScript.

כאשר ננדיר במהדר את התקן ל – **es5** ומטה, נוכל להשתמש ב – **for..of** רק עבור סוגים מסוימים הנטים – **array** ו – **string**, וכשננסח להשתמש בולאה עבור סוג נתונים אחר, זה בדוק מה שההדר יגיד לנו

Type 'number' is not an array type or a string type.

את מכוון שביקוד ה – **SJ** המומר, געשה שימוש בתוכנה **length**, שקיים רק במחזורי ומערכים.

מודולים

כברית מחדל, קוד שנכתב ב – TypeScript נמצא במרחב גלובלי. המשמעות של כך בפועל היא שם נカリ על משתנה בקובץ X, הוא יהיה זמין גם בקובץ Y, מה שועל לנורם (קרוב למדי) לתרחישים הבאים:

```

1 // fileA.ts
2 let fileName: string = "A";
3
4 // fileB.ts
5 let fileName: string = "B"; // Error! r Cannot redeclare block-scoped variable 'fileName'.
6
7 // fileC.ts
8 fileName = "C"; // Valid!
9
10

```

התנהגות זו, שמנעה מטע נקודת ההנחה שבסופו של דבר הקבצים יטענו ביחד, יוצרת מצב לא יציב שבו הכל מתרחש במרחב אחד גלובלי.. (כמו שקיים בתקן היישן).

מודולים יוצרים מרחבים מקומיים וmboidדים, והם הדרך המומלצת לכתוב TypeScript ובכלל – לכתוב Javascript. קובץ TypeScript בעצם משמש כמודול, בתנאי שהונדרו בתוכו רכיבים. ואיך מגדירים רכיבים אתם שואלים? יש לי את התשובה שאתם מחפשים!

כדי להגדיר אלמנטים (משתנה / מחלקה / ממשק / פונקציה / ...) כרכיב של המודול, נשתמש במילה

: export השמורה



```

1 // file: fileA.ts
2 export let myVar: string = "Hello World";
3
4 export function myFunc(): void {
5
6 }
7
8 export interface myInterface {
9
10}
11
12 export class myClass {
13
14}
15
16

```

כעת, האלמנטים הללו לא יהיו זמינים בקובץ **.ts**. אלא אם ניבא אותם.

ניתן גם ליצא רכיב בשם שונה מזה שניתן לו בקובץ המקור:



```

1
2 class A {
3
4 }
5
6
7 export { A as B};

```

אחרון חביב, רכיב ברירת מחדל ננדיר באמצעות המילה השמורה `default`

```

1  export default class {
2      // implementation
3  };
4
5

```

לא, אני לא בוחן אם אתם עדין איתני.. באמת לא נתתי למחלקה שם, זאת מכיוון שהשם לא רלוונטי כאשר מדובר ברכיב ברירת מחדל.
עם זאת, בהחלט אפשר לתת שם ולעתים אפילו תידרשו לכך – לדוגמה אם תיצרו מחלקה עם משתנים סטטיים.

`import`

כדי לייבא רכיבי מודול מסוים אחד לשני, נשתמש במילה השמורה `:import`

```

1  // file: fileB.ts
2
3  import { myVar } from "./fileA"
4
5

```

נקודות חשובות בנוגע לקובץ המייבא שיחסכו לכם הרבה כאב ראש:

- אין צורך למלול את הסיומת הקובץ `.ts`.
- מומלץ להשתמש במיקום יחסית (`./`) מאשר במיקום אבסולוטי, אך המהדר יודע לחפש את הקובץ גם אם לא ניתן מיקום יחסית. [קריאה נוספת](#)

כפי שניתן לראות רכיב בשם שונה, אך ניתן גם לשנות את שם הרכיב עבור הקובץ המייבא, בעת הייבוא:

```

1  import { myVar as hisVar } from "./fileA"
2
3

```

ונגלה הוכתרת: יבוא כל הרכיבים של מודול, על-ידי שימוש בטו השמור `*`:

```

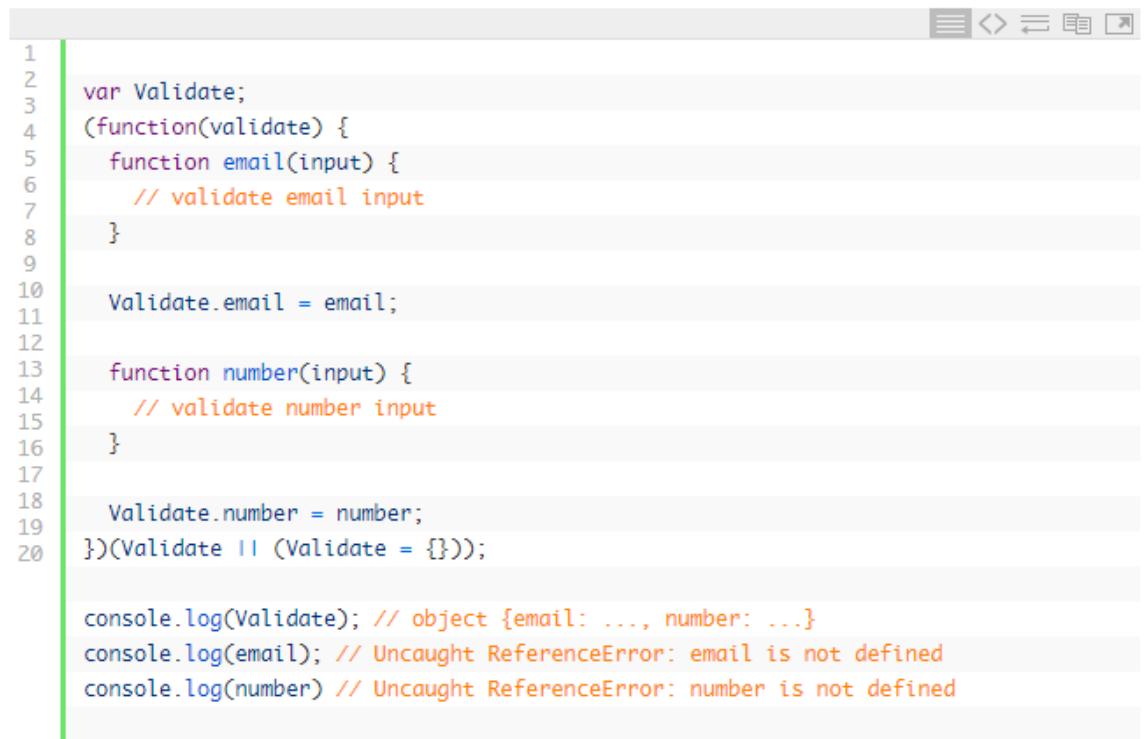
1  import * as myModule from "./fileA";
2
3  let myClass = myModule.myClass;

```

(באנגלית: Namespace)

ב – TypeScript, מתחם שמות מאפשר לאנד אלמנטים תחת שם אחד ובעצם מהוות תחליף תחבירי נכון יותר לתחביר ה – JavaScript.

אם ב – Javascript, נכתב זאת כך:



```
1 var Validate;
2 (function(validate) {
3   function email(input) {
4     // validate email input
5   }
6
7   Validate.email = email;
8
9
10  function number(input) {
11    // validate number input
12  }
13
14  Validate.number = number;
15 })(Validate || (Validate = {}));

16 console.log(Validate); // object {email: ..., number: ...}
17 console.log(email); // Uncaught ReferenceError: email is not defined
18 console.log(number) // Uncaught ReferenceError: number is not defined
```

ב – TypeScript, כל שעילינו לעשות הוא להשתמש במילה השמורה : **namespace**

```

1
2 namespace Validate {
3
4
5 }

```

כעת, כל מה שנכתב בתוך הבלוק ישתייך למתחם הנתונים `.Validate`.
בדומה למודולים, גם במתחמי נתונים ניתן לנו האפשרות להחיליט אילו אלמנטים יהיו 'ציבוריים' - נתונים
לשימוש מחוץ להנדרתת ה- `:export`, באמצעות המילה השמורה `namespace`

```

1
2 namespace Validate {
3
4   const emailRegex = /^[^@\s]+@[^\s@]+\.\[^@\s]{2,}$/;
5   export function email(input: String): Boolean | String {
6     // validate email input
7     return true;
8   }
9
10
11  const numberRegex = /[0-9].*/;
12  export function number(input: String): Boolean | String {
13    // validate number input
14    return true;
15  }
16
17 }
18

Validate.email("helloworld@masterscripter.co.il"); // Valid
Validate.emailRegex; // Error: Property 'emailRegex' does not exist on type 'typeof

```

כמובן שניתן לבצע `export` לא רק על פונקציות, אלא על הכל: משתנים, פונקציות, ממשקים, מחלקות.
הקוד יומר לנו כך בתבנית שהראתי לעיל.