

תכנות בסיסי ב- Typescript

תוכן עניינים

3.....	מבוא
4.....	פרק 1 – מבוא ל Typescript
5.....	פרק 2 – תחביר בסיסי ויצירת פרויקט Typescript
10.....	פרק 3 – פלט/קלט
14.....	פרק 4 – משתנים וקבועים
27.....	פרק 5 – תנאים
40.....	פרק 6 – לולאות
56.....	פרק 7 – פונקציות
66.....	פרק 8 – מערכים

מבוא

בספר זה אתם מתחילים את דרככם בעולם ה-TypeScript, שפה מרתקת שתמצאו בה סיפוק רב, המאפשרת תכנות בסביבת האינטרנט.

מטרת ספר זה היא להכיר את עולם התכנות החל מהיסוד. חשוב להבין שנושא הספר הינו הכרות בסיסית של עולם התכנות. לכן, בחוברת זו נתמקד אך ורק בנושאים הבסיסיים ביותר, ונציגם בצורה מדורגת ופשוטה עד כמה שניתן. במספר מקומות בחוברת זו, מצוין שאין צורך להבין חלק/מושג מסוים, אלא לכתוב אותו כמו שהוא. זאת בכדי שלא לסבך את הקורא במושגים שטרם הגיע הזמן להבינם.

אחד הדברים החשובים בלימוד תכנות הוא חזרה על החומר ותרגול רב. ספר זה מסודר כך שבסוף כל פרק ישנם תרגילים רלוונטיים. ישנו פתגם ידוע בעולם התכנות והוא ש"תכנות לומדים דרך האצבעות".

בכדי להצליח בקורס חשוב להקפיד על מספר דברים:

1. ביצוע חזרה בבית על החומר הנלמד בשיעור, וקריאת החלקים הרלוונטיים מהספר.
2. ביצוע כל תרגילי הבית.
3. לימוד עצמי של חומר, כחלק משיעורי הבית.

הקפדה על כללים אלו הנה ערובה להצלחה בקורס!

אנו מאחלים לכם לימוד פורה ומהנה, במהלכו תוכלו להיעזר במשאבים הרבים שג'ון ברייס מעמידה לרשותכם.

פרק 1 – מבוא ל Typescript

בפרק זה נלמד את הנושאים הבאים:

- Java Script
- Typescript

מהי Java Script

Java Script - שפה המאפשר יכולות תכנותיות בתוך דפדפן האינטרנט. שפות תכנות רגילות (כגון C, C++, VB, C#, Java וכד') עוברות תהליך קומפילציה. תהליך הקומפילציה הופך את הקוד שנכתב בשפת התכנות לקובץ הרצה (exe), המכיל קוד מכונה (Native Code) אותו המחשב מסוגל להבין. קובץ זה ניתן להעביר / להעתיק ממחשב למחשב ולהריץ אותו. התוכנה שמריצה את קובצי ה-exe היא מערכת ההפעלה. שפת תסריט (script) בניגוד לשפות תכנות רגילות, לא עוברת תהליך קומפילציה אשר הופך אותן לקובץ הרצה (exe), אלא מופעלות בתוך סביבה אחרת. את הקוד שנכתב בשפת Java Script יריץ הדפדפן.



הערה: שפת Java Script יכולה לשמש גם בסביבות אחרות שאינן הדפדפן, נושא זה אינו נפוץ ואינו כלול מסגרת חוברת זו.

את שפת Java Script כותבים בתוך דף ה-HTML והיא חלק ממנו. מטרתה של שפת Java Script היא לספק מימד תכנותי לדפי ה-HTML. HTML הינה שפה אשר ניתן באמצעותה להגדיר את עיצוב הדף בלבד, ולא ניתן לבצע באמצעותה פעולות תכנותיות כגון: חישובים, תפריטים נפתחים, בדיקת קלט מהמשתמש וכד'. כאן בדיוק נכנסת Java Script בכך שהיא מאפשרת יכולות אלו ועוד.

מהי Typescript

Java Script כשפה עוברת שינויים רבים בשנים האחרונות. השימוש ב-Java script הפך לכה נפוץ בגלל השימוש המוגבר והמוגדל ברשת האינטרנט לשפת הפיתוח הנפוצה בעולם. כחלק מגידול השימוש בשפה על הצורך להוסיף יכולות רבות לשפה לרבות יכולות מתקדמות הקשורות לעולם ה"תכנות מונחה עצמים". Java script בפני עצמה תתמוך ביכולות רבות נוספות בשנים הקרובות, מפני שמי שמריץ את הקוד שנכתב ב-JavaScript הוא הדפדפן עצמו נוצר מצב שכל דפדפן תומך בגרסה שונה של Java script דבר המסבך את המפתח בבחירת הגרסה הנכונה לפיתוח.

Microsoft זיהתה את הצורך ופיתחה את Typescript שהיא הרחבה ל-JavaScript שמוסיפה לשפה את כל מה שהיה חסר לה. Typescript מאפשרת לנו לכתוב קוד אחיד המכיל את כל יתרונות ה-Java script ומוכן כבר היום ליכולות שיתמכו ב-Java script בעתיד.

החיסרון הוא שלא ניתן להריץ היום Typescript בכל הדפדפנים. כאן נכנס לסיפור ה-Type script compiler (TSC) או בעברית – "המהדר". ה-Type script compiler יודע לקחת את הקוד שכתבנו ב-TypeScript ולהמיר אותו ל-Java script נקי גם בגרסאות הכי נמוכות שלו (ניתן לבחור את הגרסה המבוקשת). בצורה זו ניתן לכתוב מערכת ב-TypeScript ולהיות בטוח שהקוד ירוץ בצורה תקינה בכל הדפדפנים ובעתיד כאשר כל הדפדפנים יתמכו בגרסאות המקדמות של Java script נוכל רק לשנות את הגדרת ה-TypeScript compiler ולקבל את ההמרה לגרסה החדשה והעדכנית ביותר, ושוב, כך אנחנו כותבים קוד היום שכבר מוכן לאיך שתיראה השפה העוד מספר שנים.

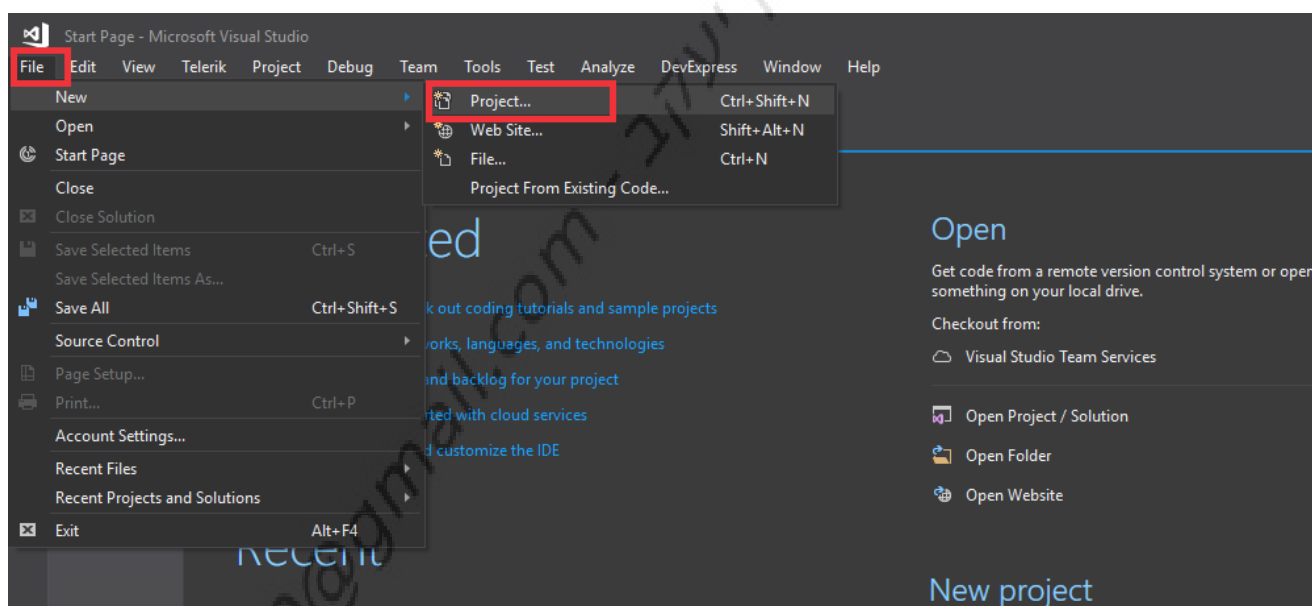
פרק 2 – תחביר בסיסי ויצירת פרויקט Typescript

- בפרק זה נלמד את הנושאים הבאים:
- יצירת פרויקט Typescript
 - תחביר בסיסי

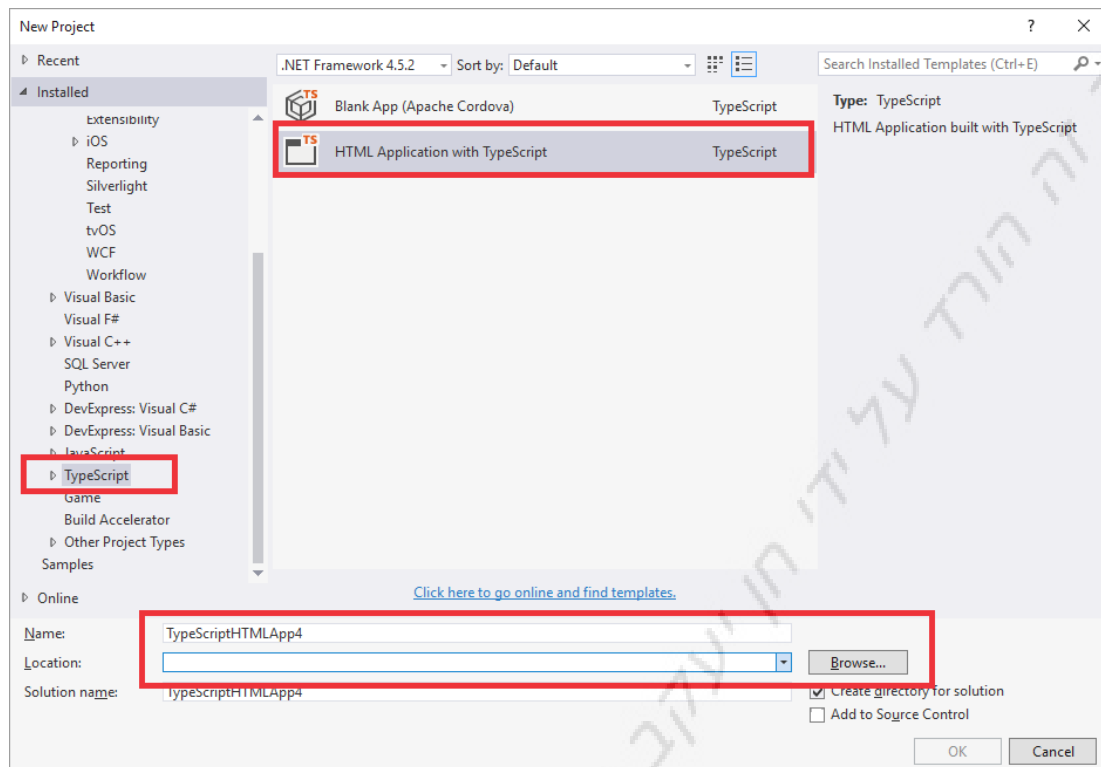
סביבת העבודה שבה נעבוד/נלמד הקורס זה תהיה Visual Studio. ניתן לכתוב Typescript גם בסביבת עבודה אחרת.

יצירת פרויקט חדש

1. על מנת לייצר פרויקט נפתח פרויקט חדש באמצעות File - > New Project.

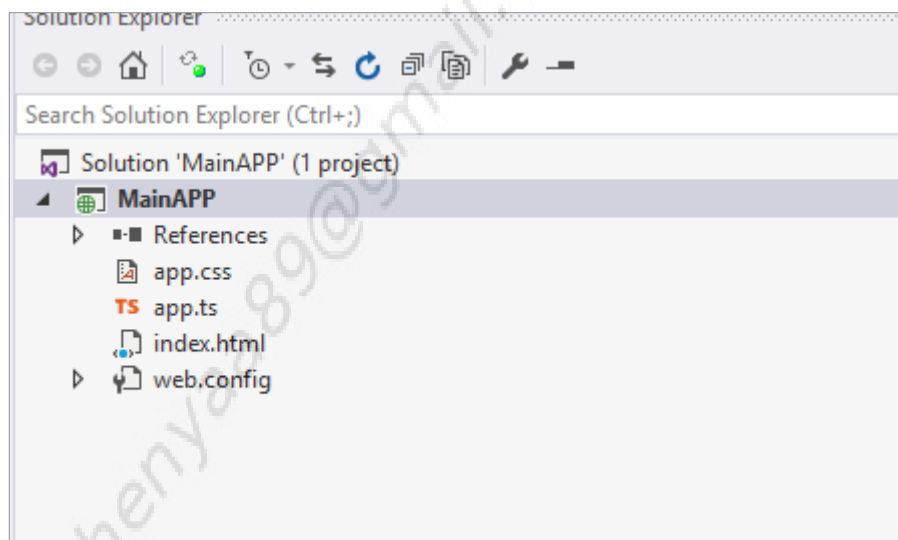


2. במסך שיפתח יש לבחור בתפריט מימין ולבחור מימין Html Application with TypeScript.

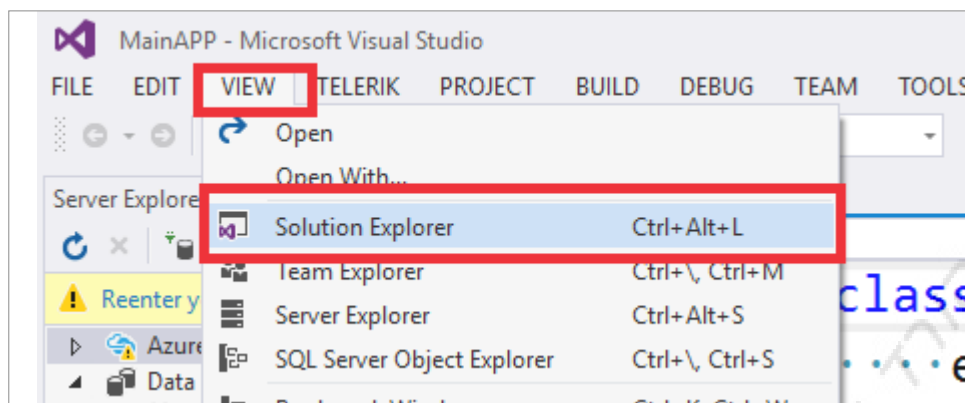


3. בתיבה התחתונה יש לציין את שם הפרויקט ואת המיקום שבו הפרויקט יוצר.

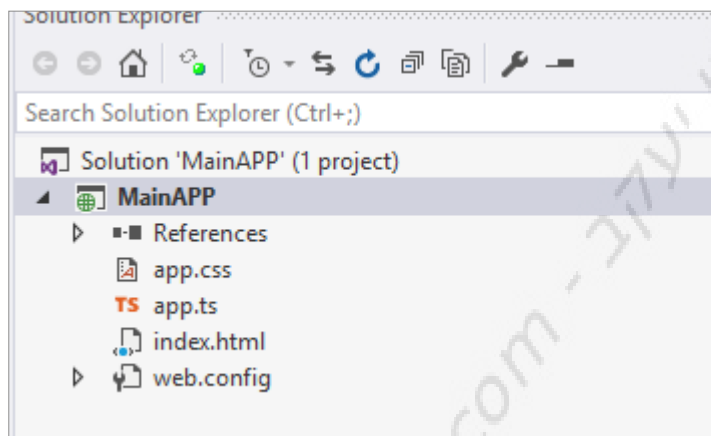
לאחר שלחצנו על אישור יפתח פרויקט חדש. אם נסתכל על ה - Solution Explorer נקבל את התמונה הבאה :



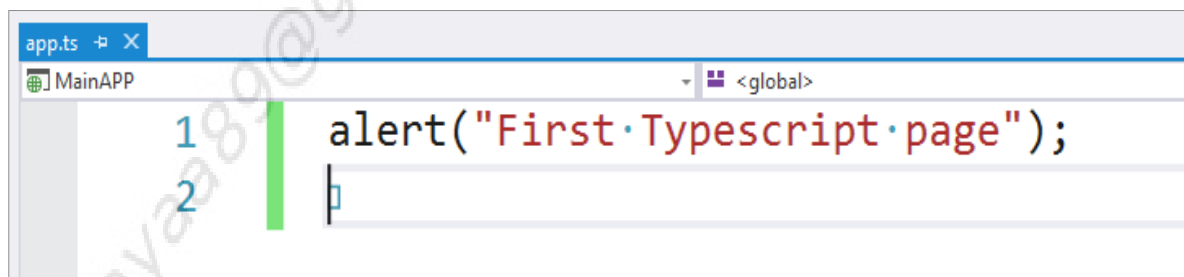
במידה ולא מוצג ה - Solution explorer ניתן לפתוח אותו באמצעות View - > Solution explorer



אם נסתכל שוב על ה - Solution Explorer נראה שם קובץ בשם app.ts. זה קובץ העבודה הראשי שלנו.

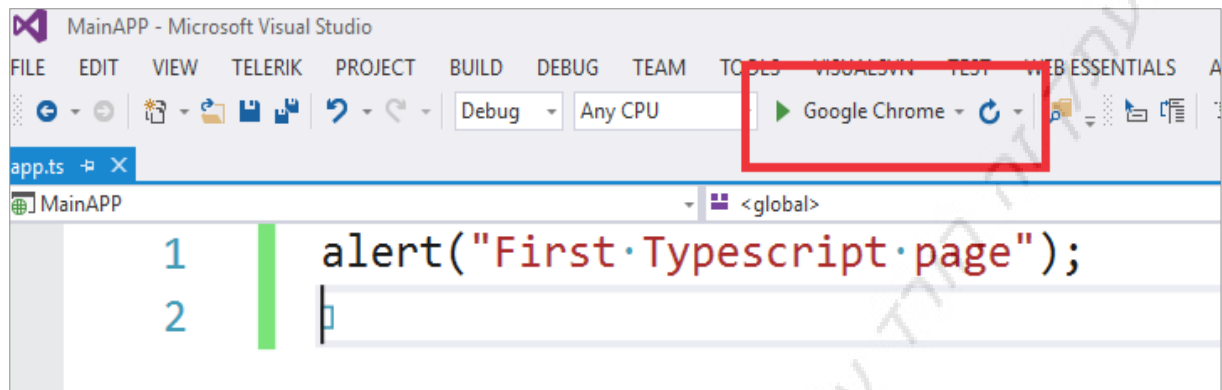


4. בשלב זה נמחק את הקוד שנוצר שם באופן אוטומטי ונכתוב קוד מ-0. נכתוב את הפקודה הבאה בתוך קובץ ה app.ts.



קובץ ts הוא קובץ Typescript וכמו שלמדנו יש להמיר את קובץ זה לקובץ Javascript רגיל על מנת שהדפדפן יוכל להריץ אותו.

5. נריץ תהליך קומפילציה וריצה של הקוד באמצעות לחיצה על ה - Play ב - Tool bar למעלה או באמצעות F5.



ברגע שנריץ את קוד הדפדפן יפתח ונקבל את ההודעה הבאה:



תפקידה של הפקודה alert להציג תיבת הודעה עם הטקסט הכתוב בגרשיים.

Syntax (תחביר) בסיסי

- כל פקודה שנכתוב ב- Typescript צריכה להסתיים בתו נקודה-פסיק (;). למרות שהדף יעבוד גם ללא כתיבת נקודה-פסיק, מומלץ מאד לכתוב כך מטעמי כתיבה נכונה.
- שפת Typescript מבחינה בין אותיות קטנות לאותיות ראשיות (Case Sensitive), ולכן חשוב להקפיד על Upper Case ועל Lower Case.

לדוגמא אם נכתוב את הפקודה הקודמת כך (האות A גדולה) נקבל שגיאה:

Alert("First Typescript page");

- **תיעוד הקוד** - מומלץ לכתוב הערות בקוד התוכנית ע"מ לזכור "מה רציתי לעשות". הערות אלו אינן חלק מהתוכנית והדפדפן יתעלם מהן. ישנן 2 אפשרויות לסימון הערות:
 1. סימון הערה על פני שורה אחת (או חלק משורה) מתבצעת באמצעות שני קווים נטויים (//). ההערה מתחילה מיד לאחר 2 הקווים הנטויים ועד לסוף השורה.
 2. לסימון הערה על פני מספר שורות יש להתחיל עם (/*) ולסיים עם (*//). ניתן גם לסמן כך חלק באמצע שורה.

דוגמא להערות:

```
//This command show's alert box  
alert("First Typescript page");  
/*  
This is  
documentation  
on multiple rows  
*/
```

מומלץ מאד להקפיד על אינדנטציות/הזחות (טאבים), ולא לכתוב בצורה בה כל השורות יתחילו מאותו מקום.



תרגיל:



- ✓ בנה תכנית אשר תציג את שמך על המסך בתוך תיבת הודעה.
- ✓ הפעל את תכנית.
- ✓ הוסף הערות לקוד הנ"ל.
- ✓ נסה לשנות את גודל אחת האותיות של הפקודה לאות גדולה ובדוק מה קורה.

פרק 3 – פלט/קלט

בפרק זה נלמד את הנושאים הבאים:

- הדפסת פלט למסך
- Text Formatting
- Numeric Formatting
- קבלת קלט מהמשתמש
- תווים מיוחדים להדפסה

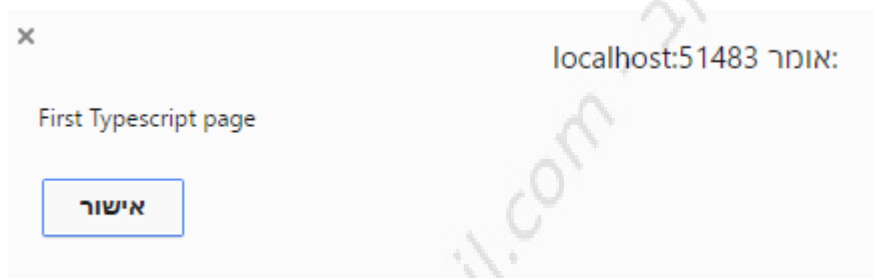
הדפסת פלט למסך

ישנן מספר מתודות (פקודות) המאפשרות הדפסת פלט למסך:

- על מנת להציג הודעה "קופצת" נתן להשתמש בפקודת alert בצורה הבאה :

```
alert("Write here what that you want to see");
```

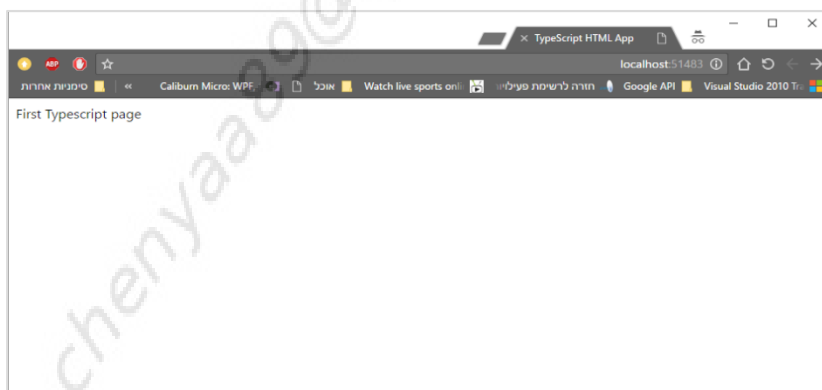
פלט:



- הדפסה לתוך הדף עצמו באמצעות document.write:

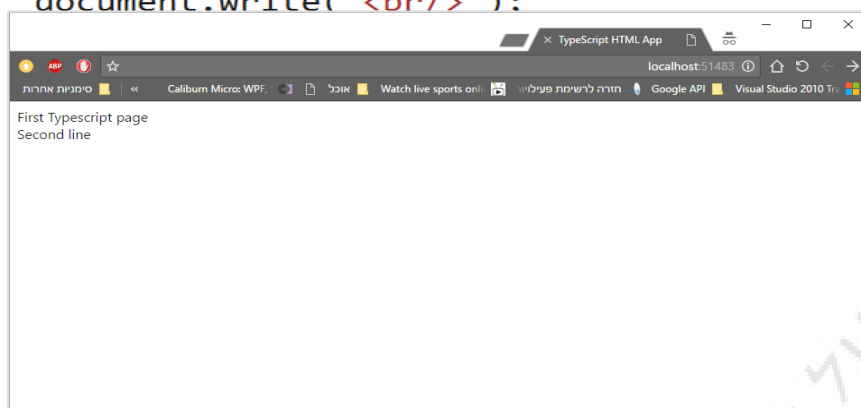
```
document.write("First Typescript page");
```

תוצאת הדף :



על מנת לבצע ירידת שורה נשתמש בכתיבה מיוחדת באמצעות `document.write`.

```
document.write("First·Typescript·page");  
document.write("<br/>");
```



השתמשנו ב"br" על מנת לרדת שורה.

תוצאה :

Text Formatting

בנוסף להדפסה של טקסט על המסך, ניתן גם "לעצב" את הפלט המודפס, תוך כדי שימוש בסמנים מיוחדים הנקראים: Markers.

בכדי לעצב את הפלט באמצעות הסמנים, יש להטמיע את הסמנים בתוך משפט הפלט. כאשר המחשב יבצע את פקודת הפלט, הוא יחליף את הסמנים בערכים המתאימים.



נעשה שימוש בגרש מיוחד במקום גרשיים, ניתן לראות בתמונה את מיקומו במקלדת

על מנת להפריד בין ערכים לטקסטים נעשה שימוש ב \$ וסוגריים מסולסלים כמו בדוגמא הבאה.

```
document.write(' The Sum of ${12} and ${23} is ${12 + 23}.');
```

הפלט:

The Sum of 12 and 23 is 35

קבלת קלט מהמשתמש

על מנת שנוכל לנהל "דו-שיח" עם המשתמש (ה-User), עלינו לאפשר קבלת מידע מהמשתמש. לאחר קליטת המידע המשתמש, נכניס את המידע לתוך מקום בזיכרון התוכנית, בכדי שנוכל להשתמש בו בהמשך. מקום זה בזיכרון נקרא משתנה. בפרק הבא נרחיב על נושא המשתנים, אך כעת כדי שנוכל לקלוט מידע מהמשתמש, נעשה שימוש במשתנה ללא הסבר מורחב.

```
var name: string;
name = prompt("Enter your name ");
document.write(name);
```

בדוגמא הנ"ל:

בשורה הראשונה ישנה הצהרה על משתנה בשם name. בשורה השנייה של התוכנית, מתבצעת הדפסה של הודעה למשתמש המבקשת ממנו לכתוב את שמו. בשורה השלישית התוכנית ממתינה עד שהמשתמש יכניס מידע. ברגע שהמשתמש ילחץ על לחצן ה-Enter המידע יכנס לתוך המשתנה name. בשורה האחרונה התוכנית מדפיסה את הערך שהמשתמש הקליד. במקרה זה התוכנית תדפיס את שמו של המשתמש.

שים לב! ניתן לראות שבתוכנית זו ההדפסה בשורה 3 היא ללא גרשיים מסביב ל-**name**, מה שאומר שיודפס תוכן המשתנה name (שזה השם שהמשתמש הכניס) ולא המילה name. דוגמא נוספת:

```
var fName: string;
var lName: string;
fName = prompt("Enter your first name ");
lName = prompt("Enter your last name ");
document.write(`Your name is ${fName} ${lName}`);
```

בדוגמא הנ"ל:

אנו רואים הצהרה על 2 משתנים (שורות 1,2). לאחר מכן הדפסת הודעה למשתמש להכניס את שמו הפרטי וקליטת הערך לתוך המשתנה **fName**. לאחר מכן הדפסת הודעה למשתמש להכניס את שם משפחתו וקליטת הערך לתוך המשתנה **lName**. לאחר מכן הדפסת השם הפרטי ושם המשפחה באותה שורה עם רווח ביניהם.

תרגילים:

1. קלוט שני ערכים והצג אותם.
2. קלוט שני ערכים והצג אותם בסדר הפוך מסדר קליטתם.
3. קלוט שני ערכים והצג את השני שנקלט.
4. קלוט שני ערכים והצג את הראשון שנקלט.
5. קלוט ערך והצג אותו שלוש פעמים.
6. קלוט ערך והצג אותו שלוש פעמים באמצעות פקודת הדפסה אחת.
7. קלוט שני נתונים למשתנה אחד! (פעמיים רצופות) והצג את תוכן המשתנה.
מה יקרה לערך הראשון שנקלט לתוך המשתנה?
8. הדפס את שמך הפרטי ושם משפחתך ב- 2 שורות נפרדות. יש להשתמש בפקודת הדפסה אחת בלבד.

פרק 4 – משתנים וקבועים

בפרק זה נלמד את הנושאים הבאים:

- מהו משתנה?
- שימוש במשתנה
- כללי מתן שמות למשתנים
- טיפוסים משתנים מספריים
- אופרטורים מתמטיים על משתנים
- מחרוזות
- קליטת ערך לתוך משתנה מספרי
- קבועים
- קיצורים בהצהרה על משתנים
- קיצורים בשימוש במשתנים
- חובת אתחול משתנים וחובת שימוש במשתנים
- העברת ערך בין משתנים מטיפוסים שונים
- יצירת מספרים אקראיים

מהו משתנה?

משתנה – מקום בזיכרון המחשב המשמש לאחסון של מידע במהלך התוכנית. המידע שהמשתנה מכיל יכול להשתנות במהלך התוכנית (לכן הוא נקרא משתנה).

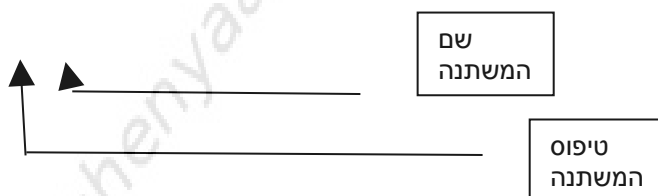
הגדרת המשתנה מורכבת מ- 2 חלקים:

1. טיפוס המשתנה – מתאר את סוג הנתונים שניתן להכיל במשתנה. טיפוס המשתנה מעיד גם על גודלו בזיכרון. לכל סוג של מידע יש את הטיפוס המתאים לו. כאשר ננסה להכניס מידע מטיפוס אחד למשתנה מטיפוס אחר יתכן אובדן מידע.
2. שם המשתנה – מאפשר לפנות דרך קוד התכנות למידע המאוחסן במשתנה.

שימוש במשתנה

על מנת להשתמש במשתנה יש להצהיר עליו. ההצהרה חייבת להיעשות לפני השימוש הראשון במשתנה.

הצהרה על משתנה:
var x: number;



כאשר מצהירים על משתנה במהלך התוכנית, מוקצה מקום בזיכרון המחשב ושומרת מקום זה לתוכנית (שביצעה את ההצהרה). הגודל של המקום המוקצה בזיכרון, נקבע לפי טיפוס המשתנה. כדי שנוכל לגשת למקום המוקצה בזיכרון, יש להשתמש בשם המשתנה.

- ישנם שני שימושים (עיקריים) שניתן לבצע על משתנה:
1. הכנסת מידע למשתנה (השמה).
 2. קבלת מידע המאוחסן במשתנה.

להלן דוגמא אותה ננתח:

```
var x: number;
x = 5;
document.Write(x);
```

בדוגמא זו אנו רואים הצהרה על משתנה בשם x מטיפוס `number`. לתוך משתנה זה נוכל להכניס אך ורק ערכים מספריים מפני שטיפוס `number` מוגדר כך (בהמשך נלמד טיפוסים משתנים נוספים).

הכנסת מידע למשתנה:

בכדי להכניס מידע לתוך משתנה, יש להשתמש באופרטור ההשמה (=). כאשר משתמשים באופרטור ההשמה, שם המשתנה צריך להיות בצד שמאל של סימן השווה (=), כך שהתוצאה של כל מה שבצד ימין לסימן השווה (=), תיכנס לתוך המשתנה. בדוגמא הנ"ל הערך 5 יושם בתוך המשתנה x .

שים לב! פעולת ההשמה אינה השוואה מתמטית.

הדפסת המידע המאוחסן במשתנה:

בכדי להשתמש במידע המאוחסן במשתנה יש לכתוב את שמו ללא גרשיים. לכן, במקרה זה אנו נבקש את המידע, את הערך, שנמצא במשתנה x , ולאחר שנקבל אותו הוא יודפס. בדוגמא שלנו, יודפס הערך 5.

```
document.Write(x);
```

פלט:
5

דוגמא נוספת:

הכרזה על שלושה משתנים.

כלל המשתנים הם מטיפוס `number` ושמותיהם הם a, b, c

```
var a : number;
var b : number;
var c : number;
```

השמה של ערכים בתוך המשתנים. במשתנה c תהיה תוצאת החיבור
של שני הערכים שבתוך a ו- b .

```
a = 4;
b = 7;
c = a + b;
```

במקרה זה, תוצאת החישוב $a + b$ ($11 = 7 + 4$) תאוחסן במשתנה c .

שים לב! לא ניתן לבצע את אותה הפעולה בדרך הבאה:

```
a = 4;
b = 7;
a + b = c;
```

הסיבה שבגינה לא ניתן לבצע את אותה הפעולה בדרך הבאה, טמונה באופן בו פעולת ההשמה מתבצעת. כאמור, כאשר אנו מבצעים השמה, שם המשתנה צריך להיות בצד שמאל של אופרטור ההשמה (=), והביטוי אותו רוצים להציב במשתנה יהיה בצד ימין של הסימן. במקרה זה המחשב ייקח את הערך של המשתנה c וינסה לאחסן ערך זה ב-a + b. כיוון ש-a + b אינו מקום שבו ניתן לאחסן ערכים, המחשב יודיע לנו ע"י הודעת שגיאה שהדבר לא ניתן לביצוע.

דוגמא נוספת:

```
// הכרזה על משתנה מסוג number. שמו של המשתנה הוא a, וכבר
// בשורה זו מתבצעת השמה של הערך 5 לתוך המשתנה.
var a : number = 5;

// קריאת הערך 5 מתוך a, חישוב, והשמה מחדש.
a = a + 1;
```

תוצאת החישוב a + 1 (6 = 1 + 5) תאוחסן במשתנה a ועכשיו ערכו של a יהיה 6. מכאן אנו מבינים שקודם מתבצע כל החישוב שבצד ימין של המשוואה, ורק לאחר סיום החישוב התוצאה תיכנס לצד שמאל של השווה (=).

כללי מתן שמות למשתנים

כללי חובה:

ישנם מספר כללי חובה לגבי מתן שמות למשתנים במידה ולא נעמוד בכללים אלו התוכנית לא תעבור קומפילציה.

- שם משתנה יכול להכיל ספרות, אותיות לועזיות (אנגלית) וקו תחתון (_) בלבד.
- שם משתנה חייב להתחיל באות לועזית (אנגלית) או בקו תחתון (_).
- שם משתנה אינו יכול להיות שם של מילה שמורה (כמו number).
- לא ניתן לתת אותו שם ל-2 משתנים שונים (בתוך אותו בלוק).

כללי רשות:

ישנם מספר כללים המומלצים מאד. במידה ולא נעמוד בכללים אלו התוכנית תפעל כראוי, אך תהיה קשה יותר לקריאה ולתחזוקה.

- יש לתת למשתנים שמות משמעותיים, שיקלו על הקורא להבין מה מטרת המשתנה ומה המשתנה אמור להכיל. למשל, שם משתנה האמור להכיל את שמו הפרטי של המשתמש יקרא firstName ולא x.

- האות הראשונה של המשתנה תהיה קטנה. משתנה המורכב משתי מילים או יותר יתחיל באות קטנה ותחילת כל מילה נוספת תהיה באות גדולה. לדוגמא: number, myNumber

- הכרזה מראש על משתנים. כאשר עובדים עם משתנים, נהוג להכריז על המשתנים בראש הקוד, בצורה מרוכזת. בחלק ניכר מהמקרים, בצמוד להכרזת המשתנים, יהיו הערות אשר יפרטו את תפקידו של כל משתנה.

אופרטורים מתמטיים על משתנים

לרוב אנחנו נשתמש במשתנים מספריים, כאשר אנחנו נרצה לשמור מידע מספרי (נומרי), ולבצע פעולות חשבוניות על הערכים המאוחסנים במשתנים. בכדי שה- Compiler יבין שברצוננו לבצע פעולות חשבוניות, עלינו להשתמש באופרטורים, בסימנים מיוחדים. עבור כל פעולה חשבונית, ישנו אופרטור מתאים:

אופרטור	פעולה
+	Addition - חיבור
-	Subtraction - חיסור
*	Multiplication - כפל
/	Division - חילוק
%	Modulus - חילוק ושארית

שים לב! ניתן להפעיל אופרטורים מתמטיים על משתנים מספריים בלבד.

דוגמא:

```
var x : number;
var y : number;
var z : number;
```

```
x = 55;
y = 17;
```

```
z = x + y;
alert(z);
```

```
z = x - y;
alert(z);
```

```
z = x / y;
alert(z);
```

```
z = x * y;
alert(z);
```

```
z = x % y;
alert(z);
```

פלט:

```
72
38
3.235294117647059
935
4
```

קדימות אופרטורים מתמטיים:

כמו בחשבון יש קדימות לאופרטורים %, /, *, על פני האופרטורים +, -
לדוגמא:

בקטע הקוד הבא יכנס לתוך המשתנה z הערך 12, מכיוון שקודם בוצעה פעולת החילוק ($2/y$ שזה 2) ולאחר מכן התוצאה נוספה לערך שבתוך x (שזה 10).

```
x = 10;
y = 4;
z = x + y / 2;
```

ניתן לשנות את קדימות האופרטורים באמצעות סוגריים ולכן במקרה הבא הערך שיכנס לתוך z יהיה 6 (קודם $x + y$ ורק לאחר מכן חלוקת התוצאה ב 2):

```
x = 10;
y = 3;
z = (x + y) / 2;
```

מחרוזות

מחרוזת - רצף של תווים מכל סוג שהוא (אותיות, מספרים ותווים אחרים).

בכדי לאחסן מחרוזת בזיכרון, יש להשתמש במשתנה מטיפוס String. כבר הספקנו להשתמש בטיפוס זה, בכדי לקבל קלט מהמשתמש וכעת נרחיב עליו. כאשר במהלך הקוד נכניס ערך לתוך משתנה, כך שהערך המוכנס יהיה עטוף בגרשיים, למעשה קבענו שהטיפוס של המשתנה הוא String.

לדוגמא:

```
var s : string;
s = "My name is avi, and my age is 5";
```

בדוגמא זו אנו רואים שניתן להכניס כל תו לתוך מחרוזת.

ניתן להשתמש גם ב 2 גרשים (פעמיים גרש בודד):

```
var s : string;
s = 'My name is avi, and my age is 5';
```

דוגמא זו שקולה לדוגמא הקודמת.

היכולת להשתמש בשתי האפשרויות מאפשרת להכניס גרש או גרשיים כחלק מהמחרוזת ללא צורך בשימוש בתווים המיוחדים (אלו שמתחילים ב \). כאשר נעטוף את המחרוזת בגרשיים משני הצדדים נוכל לכתוב גרש בודד כחלק מהמחרוזת, וכאשר נעטוף את המחרוזת בגרש משני הצדדים נוכל לכתוב גרשיים כחלק מהמחרוזת.

לדוגמא:

```
var s : string;
s = "I'm doing something";
```

במקום לכתוב:

```
var s : string;
s = 'I'm doing something';
```

גם כאשר מציגים טקסט על תיבת alert או תיבת prompt, אנו עוטפים אותו בגרשיים, שזה אומר שאנו מציגים String.
הערה: גם כאשר מציגים ערך מספרי הוא עובר המרה ל - string, דבר המתבצע אוטומטית ע"י הפקודה alert (או prompt).
ניתן לאפס משתנה מחרוזת, ע"י הכנסת מחרוזת ריקה למשתנה:
לאחר קטע הקוד הבא הערך במשתנה s יהיה מחרוזת ריקה.

```
var s : string;
s = "aaaaa";
s = "";
```

שרשור (חיבור) מחרוזות:

ניתן לשרשר (לחבר) בין 2 מחרוזות (או יותר) ולהפוך אותן למחרוזת אחת. בכדי לבצע זאת נשתמש באופרטור השרשור (+).

```
var s1 : string;
var s2 : string;
```

```
s1 = "acb";
s2 = s1 + "def";
```

לאחר ביצוע פקודות אלו, יהיה בתוך s2 המחרוזת "abcdef".

יש לשים לב שאופרטור השרשור שונה מאופרטור החיבור, למרות שאת שניהם כותבים אותו הדבר (+):

```
var s1 : string;
var s2 : string;
```

```
s1 = "1";
s2 = s1 + "2";
```

בדוגמא זו הערך שיהיה בתוך s2 הוא "12" ולא תוצאת החיבור בין 1 ל- 2.

דבר זה מעיד שחשוב מאד לקבוע את טיפוס המשתנה בצורה נכונה, אחרת אנו עלולים לקבל תוצאות לא רצויות.

קליטת ערך לתוך משתנה מספרי

ע"מ לקלוט ערך מהמשתמש לתוך משתנה עלינו להשתמש בפונקציה **prompt**. הפונקציה הנ"ל מוגדרת כך שתחזיר את הקלט מהמשתמש כטיפוס מסוג String. טיפוס זה יכול להכיל בתוכו מחרוזת שזה אומר כל תו הנמצא על המקלדת (אותיות, ספרות, סימנים וכו').
לא ניתן באמצעות הפונקציה prompt לקבל את הקלט לתוך משתנה שאינו מטיפוס String.

בכדי שנוכל לבצע פעולות חשבון על משתנה, הוא חייב להיות מטיפוס מספרי, ומכיוון שכך, יש להעביר את הערך הנמצא בתוך משתנה מסוג String (שחוזר מהפונקציה prompt), לתוך משתנה אחר מטיפוס מספרי.

על מנת לבצע פעולה זו, יש להשתמש באחת משתי הפונקציות הבאות:

parseInt – עבור מספרים שלמים

parseFloat – עבור מספרים עשרוניים

פונקציות אלו מקבלות בסוגריים משתנה מסוג String, ומחזירות אותו בפורמט מספרי.

לדוגמא:

```
var str;
str = "10";
var x;
x = parseInt(str);
```

הערך הנמצא בתוך המשתנה **str** (מטיפוס String) יעבור המרה לערך מספרי, ויכנס לתוך המשתנה x שיוגדר מטיפוס Number.

אותו הדבר בדיוק יש לבצע לגבי הפונקציה parseFloat.

לדוגמא:

```
var str;
str = "5.5";
var fl;
fl = parseFloat(str);
```

חשוב לדעת שבמידה ובתוך המשתנה **str** יהיה ערך שאינו יכול להפוך לערך מספרי (לדוגמא: "abc" או מחרוזת ריקה), יכנס לתוך השתנה ערך המכונה (NaN) Not a Number. בהמשך נלמד איך ניתן לבדוק האם הערך שהמשתנה הזין הוא אכן מספרי.

ננתח את הדוגמא הבאה:

```
var input : string;
var x : number;
input = prompt("Enter number:", "");
x = parseInt(input);
x = x + x;
alert(x);
```

בשורות 1 ו-2, אנו רואים הצהרה על משתנים (input, x).
בשורה 3, מתבצעת הקליטה מהמשתמש לתוך המשתנה input, כזכור ניתן לקלוט מהמשתמש רק לתוך משתנה מטיפוס string.
בשורה 4 מתבצעת ההמרה של המשתנה input, לערך מספרי, כאשר התוצאה (הערך המספרי) תיכנס לתוך המשתנה x. במידה והמשתמש הכניס ערך שאינו מספרי, הערך בתוך x יהיה NaN.
בשורה 5 ישנו חישוב, בו המספר הנמצא בתוך x מוכפל ב-2 והתוצאה מוכנסת חזרה לתוך המשתנה x.
לא ניתן לבצע חישוב זה במידה ו-x לא היה משתנה מספרי (Number) מפני שאופרטורים מתמטיים יכולים לפעול רק על משנים מספריים.
ובשורה 6 מוצגת התוצאה למשתמש.

קבועים

ניתן להסתכל על קבוע כעל תא בזיכרון בו אנו יכולים לאחסן מידע כמו שמשתנה גם הוא תא בזיכרון, אך עם קבוע אחסון המידע הוא חד-פעמי. ברגע שקבענו את ערכו של הקבוע, לא ניתן יותר לשנות אותו לכל אורך התוכנית. לעיתים נרצה במהלך התוכנית להשתמש בערך קבוע, שלא אמור להשתנות במהלך התוכנית.
לדוגמא: מספר הימים שיש בשבוע, הערך פאי בגיאומטריה וכו'.

מספר כללים לגבי קבועים:

- חובה לתת לקבוע את ערכו מיד בשורת ההגדרה
- לא ניתן לשנות ערך של קבוע לכל אורך התוכנית
- השימוש בקבוע הינו בדיוק כמו שימוש במשתנה (למעט העובדה שלא ניתן לשנות אותו)
- בכדי שהיה קל לזהות במהלך התוכנית שמדובר בקבוע, נהוג ששמו יהיה באותיות גדולות ובין המילים יהיה קו תחתון () לדוגמא: DAYS_IN_WEEK

בכדי להגדיר ערך קבוע נשתמש במילה השמורה **const** לפני ההגדרה, כאשר בהגדרה אנו נכלול גם את טיפוס המידע שנשמר בקבוע, ואת שמו. לדוגמא:

הגדרת קבוע בשם PI מטיפוס double (טיפוס מספרי, לא שלם).
`const PI : number = 3.14;`

לא ניתן להגדיר קבוע בלי לאתחל אותו מיד בערך:

שגיאה. דרוש ערך.

`const PI: number;`

לא ניתן לשנות ערך של קבוע לאחר הגדרתו:

`const double PI = 3.14;`

שגיאה. לא ניתן לשנות ערכו של קבוע.

`const PI: number = 3.14;`
`PI = 1;`

שימוש בקבוע:

```
const PI: number = 3.14;

var circleArea: number;
var radius: number;
var str: string;

str = prompt("Enter radius:");
radius = parseFloat(str);
circleArea = PI * radius * radius;
document.write(`The Area is ${circleArea}`);
```

תוכנית זו קולטת מהמשתמש רדיוס מעגל ומחשבת את שטחו. ניתן לראות בשורה הראשונה הצהרה על הקבוע PI, ובשורה שלפני האחרונה חישוב של השטח באמצעות הקבוע PI כאשר השימוש ב-PI הינו כמו שימוש במשתנה רגיל.

לשימוש בקבועים יתרונות רבים, ושימוש מושכל בהם יביא לתוכנית טובה/קריאה יותר.

יתרון חשוב הוא עדכון התוכנית בקלות – במידה והשתמשנו בערך מסוים במספר מקומות בתוכנית ונרצה לשנות אותו, נאלץ לשנות בכל מקום בקוד. לעומת זאת אם נגדיר אותו כקבוע, מספיק לשנות רק בשורת ההצהרה. לכן, במידה וישנו ערך, המופיע יותר מפעם אחת בקוד, מומלץ מאד להגדיר אותו כקבוע.

קיצורים בשימוש במשתנים

אופרטורים לקיצור השמה:
נשתמש באופרטורים אלו כאשר נרצה להכניס ערך לתוך משתנה, והערך החדש הינו שינוי של הערך הקיים כבר במשתנה.
לדוגמא: הוספה של 2 לערך הקיים, חילוק ב-3 של הערך הקיים וכו'.

האופרטורים בהם נשתמש:

```
+=
-=
*=
/=
%=
```

לדוגמא:

הפעולה המקוצרת	הפעולה
$a += 5;$	$a = a + 5;$
$a -= 6;$	$a = a - 6;$
$a *= 3;$	$a = a * 3;$
$a /= 2;$	$a = a / 2;$
$a \% = 4;$	$a = a \% 4;$

ניתן לבצע זאת גם עם משתנה נוסף:

הפעולה המקוצרת	הפעולה
$a += b;$	$a = a + b;$
$a -= b;$	$a = a - b;$
$a *= b;$	$a = a * b;$
$a /= b;$	$a = a / b;$
$a \% = b;$	$a = a \% b;$

בנוסף לאופרטורים הנ"ל, ישנם שני אופרטורים נוספים המאפשרים לשנות את ערכו של המשתנה ב-1 (להוסיף או להפחית):

++ - קידום באחד מוקדם או מאוחר – **Pre or Post Increment**
-- - חיסור באחד מוקדם או מאוחר – **Pre or Post Decrement**

הפעולה	Pre	Post	ניתן לכתוב גם כך
$A = a + 1;$	$++a;$	$a++;$	$a += 1;$
$A = a - 1;$	$--a;$	$a--;$	$a -= 1;$

כמו שאנו רואים בטבלה הנ"ל, ישנן שתי אפשרויות לשימוש באופרטורים אלו:

- Pre** – כתיבת האופרטור לפני שם המשתנה. כאשר נכתוב כך בפקודה מסוימת, יתבצעו הפעולות הבאות לפי הסדר בו הן מופיעות:
 - קבלת הערך של המשתנה מהזיכרון.
 - שינוי של הערך ב-1 (הוספה/החסרה).
 - שמירה של הערך המעודכן לזיכרון.
 - שימוש בערך המעודכן.
- Post** – כתיבת האופרטור לאחר שם המשתנה. כאשר נכתוב כך בפקודה מסוימת, יתבצעו הפעולות הבאות לפי הסדר בו הן מופיעות:
 - קבלת הערך של המשתנה מהזיכרון.
 - שימוש בערך של המשתנה.
 - שינוי של הערך ב-1 (הוספה/החסרה).
 - שמירה של הערך המעודכן לזיכרון.

לדוגמא:

בקטע הקוד הבא נעשה שימוש באופרטור Post Increment (++ לאחר שם המשתנה) כיוון שבדוגמא זו אנו משתמשים ב- Post Increment, יקרו הפעולות הבאות:

1. נקבל את הערך של num מהזיכרון. במקרה שלנו, הערך הוא 10.
2. הערך של num, 10, יושם בתור הערך של המשתנה result.
3. לערך של num יתווסף 1, ולכן הוא יהפוך מ-10 ל-11.
4. הערך המעודכן יישמר בזיכרון בתור הערך של num.

מכאן ניתן להבין שפקודת ההשמה בוצעה לפי הערך הישן של num.

```
var num : number = 10;
var result : number ;
result = num++;
alert(`num = ${num}`);
alert(`result = ${result}`);
```

הפלט:

```
num = 11
result = 10
```

לעומת זאת, בקטע הקוד הבא נעשה שימוש באופרטור Pre Increment (++ לפני שם המשתנה). כיוון שבקטע קוד זה אנו משתמשים ב- Pre Increment, יתרחשו הפעולות הבאות:

1. נקבל את ערכו של num מהזיכרון. במקרה שלנו הערך הוא 10.
2. לערך של num יתווסף 1, ולכן הוא יהפוך מ-10 ל-11.
3. נשמור את הערך המעודכן לתוך num.
4. הערך (המעודכן) של num יושם בתור הערך של result.

מכאן ניתן להבין שפקודת ההשמה בוצעה לפי הערך החדש של num.

```
var num : number = 10;
var result : number ;
result = ++num;
alert(`num = ${num}`);
alert(`result = ${result}`);
```

הפלט:

```
num = 11
result = 11
```

העברת ערך בין משתנה מספרי למשתנה מסוג מחרוזת

במהלך הפרק ראינו את הצורך בהעברת ערך בין משתנים מטיפוסים שונים, כאשר קלטנו מידע מהמשתמש לתוך משתנה מטיפוס String והעברנו אותו למשתנה מטיפוס Number, בכדי שנוכל לבצע עליו פעולות חשבון.

לעיתים נרצה לבצע את הפעולה ההפוכה, ולהעביר משתנה מטיפוס מספרי לתוך משתנה מטיפוס String, כדי שנוכל להשתמש ביכולות הקיימות רק במחרוזות (כמו שרשור). המרה מטיפוס מספרי למחרוזת מתבצעת דרך שם המשתנה ולאחריו toString (הוספת נקודה ו - toString)

לדוגמא:

```
var str : string;
var x : number = 10;
str = x.toString();
```

דרך נוספת לביצוע המרה ממספר למחרוזת היא באמצעות שרשור של מחרוזת ריקה למספר אותו רוצים להמיר, דרך זו תבצע את אותה הפעולה של הדוגמא הקודמת:

```
var str : string ;
var x : number = 10;
str = "" + x;
```

יצירת מספרים אקראיים

ניתן לבקש מהמחשב ליצור מספרים אקראיים במהלך התוכנית.

הפקודה שמבצעת זאת היא: Math.random() (פקודה זו מחוללת מספר לא שלם אקראי בין 0 ל 1 (כולל 0, לא כולל 1))

לדוגמא:

```
var n : number;
n = Math.random();
alert(n);
```

תוכנית זו תציג בכל פעם פלט אחר, בהתאם למספר שיוגדל.

בד"כ כאשר נרצה מספר אקראי, לא נסתפק במספר בין 0 ל 1 אלא נרצה טווח מספרים קצת שונה לדוגמא: מספר שלם בין 0 ל 20. לצורך כך נצטרך לבצע מספר פעולות:

```
var x : number;
x = Math.random();
x = x * 21;
x = Math.floor(x);
alert(x);
```

בשורה 1 ישנה הצהרה על משתנה בשם x

בשורה 2 יצירת המספר האקראי (בין 0 ל-1)
בשורה 3 מכפילים את המספר במספר הגבוה ביותר האפשרי + 1. פעולה זו תגרום לכך שיהיה לנו מספר בין 0 ל- 21 כולל מספר ספרות לאחר הנקודה העשרונית.
בשורה 4 יש שימוש בפקודה Math.floor – פקודה זו מקבלת מספר ומקצצת את כל מה שיש אחרי הנקודה העשרונית. לאחר שורה זו יהיה לנו מספר שלם בין 0 ל- 20 (כנדרש).

לפעמים נרצה שהמספר התחתון בטווח לא יהיה 0 אלא יותר גבוה לדוגמא: בין 60 ל- 100 (כולל).
לצורך כך נצטרך להחליף את שורה 3 בשורה הבאה:

$$x = x * (100 - 60 + 1) + 60;$$

הנוסחה אומרת:

מספר אקראי * (מספר עליון – מספר תחתון + 1) + מספר תחתון

תרגילים:

- קלוט שלושה מספרים. סכם אותם, הצג את שלושתם ואת תוצאת הסיכום.
- קלוט שלושה מספרים. הצג את שלושתם ואת הממוצע החשבוני שלהם. שים לב! הממוצע צריך להיות מדויק (עם נקודה עשרונית).
- קלוט 3 מספרים והצג את מכפלתם.
- קלוט מספר והדפס את הריבוע שלו (מכפלתו בעצמו).
- קלוט ארבעה מספרים וחשב לפי הסדר הבא:
 - הוסף את הראשון לשלישי.
 - החסר את השני מהרביעי.
 - חלק את השלישי המקורי ב- 8.
 - הכפל את הרביעי בראשון (לפני ההוספה).
 - הצג את התוצאות.
- לפתרון התרגיל מותר להוסיף משתני עזר.
- אתגר: חזור על התרגיל הקודם ללא משתני עזר כלל.
- קלוט שני משתנים A, B לאחר מכן החלף את תוכנם (השתמש במשתנה עזר).
- קלוט אורך ורוחב של חדר מלבני. הצג את שטח החדר ואת היקפו.
- קלוט קוטר ועומק של סיר והצג את הקיבולת שלו.
(שטח עיגול הוא R , $R \approx 3.14$, $\square = R^2$. הוא בדיוק $\frac{1}{2}$ מהקוטר, וקיבולת = עומק * שטח).
- קלוט אורך של סרט קולנוע בדקות והצג את אורך הסרט בשעות ודקות.
לדוגמא: קלט – 88, פלט – 1 hour(s) and 28 minute(s).
- קלוט מספר, מובטח כי הוא בן 4 ספרות לפחות. מצא מהי ספרתו הימנית ביותר והדפס אותה.
- קלוט מספר, מובטח כי הוא בן 4 ספרות לפחות. מצא מהי ספרתו השנייה מימין והדפס אותה.
- קלוט נתון תלת ספרתי והדפס את ספרת המאות (מובטח כי אינה 0).
- קלוט נתון דו ספרתי והדפס את סכום ספרותיו.
- קלוט מספר שלם בין 10 לבין 99. הפוך את סדר הספרות והצג את המספר החדש.
לדוגמא: קלט – 61, פלט – 16.
- קלוט מספר ממשי, והדפס את הספרות שלאחר הנקודה.

פרק 5 – תנאים

בפרק זה נלמד את הנושאים הבאים:

- ביטוי בוליאני
- הטיפוס Boolean
- תנאי if בסיסי
- משפט תנאי מורכב
- תנאים מקוננים
- אופרטור ! (Not)
- תנאי מקוצר
- תנאי switch
- תיבת confirm

ביטוי בוליאני (Boolean)

נתונים שני משתנים: $a = 3$, $b = 9$.
כאשר נשאל מי גדול יותר, תשובתנו המיידית תהיה b . כאשר נרצה לקבל תשובה מהמחשב, נצטרך "ללמד" אותו כיצד להחליט על תשובתו. החלטת המחשב הינה חד משמעית, "כן" או "לא". המחשב לא יכול להגיע להחלטה אחרת (שאינה כן או לא).

ביטוי בוליאני – ביטוי אשר תפקידו לבדוק ערך מסוים או יותר ולהחזיר תשובה. התשובה חייבת להיות "כן" (true) או "לא" (false).

אופרטורים המשמשים ליצירת ביטויים בוליאניים:

אופרטור	משמעות
>	גדול מ...
<	קטן מ...
==	שווה ל...
!=	שונה מ...
>=	גדול מ... או שווה ל...
<=	קטן מ... או שווה ל...

דוגמאות לביטויים בוליאניים:

בדוגמאות הבאות יש שימוש במשתנים מטיפוס Number בשמות a, b כאשר:

```
var a = 7;  
var b = 4;
```

ביטוי	תשובה
$a > b$	true
$b <= a$	true
$a > 10$	false
$b > -5$	true
$b < 4$	false
$a >= 7$	true
$a == 4$	false
$a != b$	true

הטיפוס Boolean

עד עתה הכרנו שני סוגי טיפוסים: מספר, מחרוזת.
כעת נכיר טיפוס נוסף בשם boolean. תפקידו של משתנה מטיפוס boolean הוא להכיל אחד מהערכים: true או false. משתנה מטיפוס boolean אינו יכול להכיל ערך מסוג אחר.

שימוש במשתנה מטיפוס boolean:

```
var b1 : boolean = true;
var b2 : boolean = false;
alert("b1 = " + b1);
alert("b2 = " + b2);
```

פלט:

```
b1 = true
b2 = false
```

מכיוון שמשתנה מטיפוס boolean יכול להכיל true או false, ניתן להכניס לתוך משתנה מטיפוס זה תשובה של ביטוי בוליאני.
בדוגמא הבאה אנו רואים שלתוך המשתנה b (מטיפוס boolean) תיכנס תוצאת השאלה $x \geq y$:

```
var x : number = 10, y : number = 13;
var b : boolean;
b = x >= y;
alert('The result of 'x >= y' is ${b}');
```

פלט:

```
The result of 'x >= y' is false
```

תנאי if בסיסי

עד כה, הפקודות שכתבנו בתוכנית התבצעו ברצף, מהפקודה הראשונה ועד לפקודה האחרונה. לעיתים קרובות נרצה להתנות ביצוע של קוד בקיום תנאי מוגדר. יתכן אף מצב שנרצה להתנות קטע קוד אחר באי קיומו של אותו תנאי מוגדר.

משפט תנאי if – משפט המאפשר למחשב להחליט האם לבצע קטע קוד מסוים, כאשר ההחלטה מותנית בביטוי בוליאני (Boolean).

מבנה בסיסי של תנאי בוליאני:

```
if (ביטוי בוליאני)
{
    פקודות שיתבצעו במידה
    והתנאי נכון
}
else
{
    פקודות שיתבצעו במידה
    והתנאי אינו נכון
}
```

בדוגמא הבאה ישנם שני משתנים מטיפוס Number המאותחלים בשורת ההגדרה. לאחר מכן ישנה בדיקה האם $x > y$. אם התשובה לביטוי היא true אז תודפס הודעה מתאימה, ואם false אזי תודפס גם כן הודעה מתאימה.
מכיוון שהתשובה לתנאי היא false יתבצע החלק של ה-else.

```
var x : number = 5, y : number = 8;
```

```
if(x > y)
{
    alert("x greater then y");
}
else
{
    alert("x lower then y or equals to y");
}
```

פלט:

x lower then y or equals to y

דרך נוספת לכתיבת התנאי הנ"ל:

מכיוון שמשתנה מטיפוס boolean יכול להכיל בתוכו true/false ניתן לבדוק את הערך שלו באמצעות תנאי if.

```
var x : number = 5, y : number = 8;
var b : boolean = x > y;
```

```
if(b)
{
    alert("x greater then y");
}
else
{
    alert("x lower then y or equals to y");
}
```

כאשר בכל בלוק של התנאי ישנה רק פקודה אחת ניתן לוותר על הסוגריים:

```
var x : number = 5, y : number = 8;
```

```
if(x > y)
    alert("x greater then y");
else
    alert("x lower then y or equals to y");
```

לעיתים נרצה לבצע פקודה/ות במידה והתנאי מתקיים, אך לא לבצע שום פקודה במידה והתנאי אינו מתקיים. במקרה כזה נוותר על החלק של ה-else, אשר אינו חייב להופיע בתנאי if. בדוגמא הבאה לא יודפס פלט מכיוון שאין את חלק ה-else. מה שאומר שכאשר התנאי אינו נכון לא יתבצע דבר.

```
var x : number = 5, y : number = 8;
```

```
if(x > y)
    alert("x greater then y");
```

let

let נותן לנו את האפשרות להגדרת משתנים מקומיים, גם ברמת התנאי כמו שנראה במודול זה וגם ברמת פונקציה וברמת לולאה או כל Scope שנלמד עליו בהמשך.
ניתן להגדיר את הקוד הבא :

```
let x = 2;
```

בצורה זו x הוא משתנה ברמת הScope בו הוא מוגדר אך לא מוגדר עבורו סוג (type) מסוים. ולכן יהיה מותר לנו לכתוב בשורה הבאה :

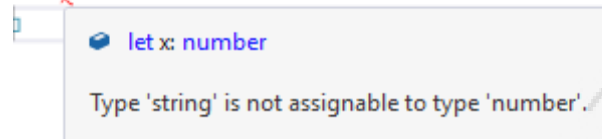
```
x = "abc";
```

אם נרצה להגדיר עבורו סוג מסוים ובכך לקבע את הסוג של אותו משתנה נשתמש ב" (נקודתיים).
לדוגמא :

```
let x: number = 2;
```

בצורה זו קיבענו את הסוג לnumber- ולכן אם ננסה לבצע השמה למשתנה לכל דבר שהוא לא number נקבל שגיאה:

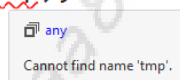
```
...let x: number = 2;
...x = "abc";
```



אם ננסה לעשות שימוש במשתנה מחוץ ל Scope שבוא הגדרנו אותו נקבל שגיאה:

```
var x: number = 10, y: number = 9;
```

```
if (x > y) {
  ...let tmp: string = "Wow";
  ...alert(tmp);
}
alert(tmp);
```



משפט תנאי מורכב

ישנם מקרים בהם לא מספיקה שאלה אחת, בכדי להחליט האם לבצע או לא לבצע פקודות מסוימות.
ניתן לשאול יותר משאלה אחת בביטוי בוליאני. לשם כך נשתמש באופרטורים הבאים:

&& - אופרטור And (וגם)
|| - אופרטור Or (או)

האופרטור **&&** (And) מייצג חיבור בין שני משפטי תנאי, כך שרק במידה ושני המשפטים מחזירים **true**, התנאי הכולל (החיבור של 2 התנאים) יחזיר **true**. כל מקרה אחר יגרום לתנאי הכולל להחזיר **false**.

האופרטור **||** (Or) מייצג חיבור בין שני משפטי תנאי, כך שמספיק שאחד התנאים יתקיים כדי שהתנאי הכולל (החיבור בין 2 התנאים) יחזיר **true**. רק במידה ושני התנאים לא מתקיימים, התנאי הכולל יחזיר **false**.

דוגמאות למשפטי תנאי מורכבים:

בדוגמאות הבאות יש שימוש במשתנים מטיפוס Number בשמות a,b,c כאשר:

```
var a : number = 7;
var b : number = 4;
var c : number = -3;
```

תשובה	ביטוי
true	$a > b \ \&\& \ b > c$
false	$b \leq a \ \&\& \ c \geq b$
true	$a > 10 \ \ b < 5$
false	$b > -5 \ \&\& \ c < -9$
false	$b < 4 \ \ c > -3$
true	$a \geq 7 \ \&\& \ c \neq a$
true	$a == c \ \ b \neq a$
true	$a < b \ \ c < b$

להלן תוכנית הקולטת מהמשתמש מספר ובודקת האם המספר הוא אחד מימות השבוע:

```
var str : string;
var dayOfWeek : number;

str = prompt("Enter day of week:", "");
dayOfWeek = parseInt(str);

if(dayOfWeek >= 1 && dayOfWeek <= 7)
    alert("Your number is well day of week");
else
    alert("Your number can't be day of week");
```

במידה והמשתמש יכניס ערך בטווח 1-7 (ימות השבוע), יודפס:

Your number is well day of week

במידה ולא יודפס:

Your number can't be day of week

טבלת התוצאות בשימוש באופרטורים ||, &&:

תנאי 1	תנאי 2	תנאי 1 && תנאי 2	תנאי 1 תנאי 2
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

שילוב יותר מ-2 תנאים במשפט תנאי:

ניתן במשפט תנאי אחד לשלב יותר מ-2 תנאים. אם נכתוב 3 תנאים כאשר ביניהם האופרטור && התנאי הכולל יחזיר true רק במידה ושלושת התנאים יתקיימו. אם נכתוב 3 תנאים כאשר ביניהם האופרטור ||, יספיק שתנאי אחד מתוך השלושה יתקיים כדי שיחזור true.

ניתן אף לשלב בין האופרטורים &&, || באותו משפט תנאי. חשוב לדעת שלאופרטור && קדימות על פני האופרטור || (כמו קדימות של כפל על-פני חיבור). ניתן לשנות את סדר הקדימויות ע"י שימוש בסוגריים (כמו בפעולות חשבון).

לכן כתיבת תנאי בצורה הבאה:

תנאי 3 || תנאי 2 && תנאי 1

זה כמו לכתוב:

(תנאי 3 || (תנאי 2 && תנאי 1)

אך לא כמו:

(תנאי 3 || תנאי 2) && תנאי 1

בדוגמאות הבאות יש שימוש במשתנים מטיפוס Number בשמות a,b,c כאשר:

var a : number = 7;

var b : number = 4;

var c : number = -3;

תשובה	ביטוי
false	a > b && b > c && b > 5
true	b <= a && c >= b c == -3
true	a > 10 b < 5 c >= 0
true	a < 10 b > -5 && c < -9
false	b < 4 c > -3 && a >= 0
false	b <= a && (c >= b c == -3)
false	(a < 10 b > -5) && c < -9
false	(b < 4 c > -3) && a >= 0

תנאים מקוננים

שאלה שנענתה בשאלה נוספת מהווה מנגנון של תנאי מקונן. פתרון בעיות מורכבות נעשה בעזרת מספר תנאים.

למשל, נקלוט שני מספרים ונציג את הגדול מביניהם. אם המספרים שווים נציג הודעה שהם שווים. כאשר נשאל שאלה אחת: האם num1 גדול מ- num2 (אם כן פתרנו את הבעיה), אך אם לא, אין זה

אומר ש- num2 הוא הגדול כי יכול להיות שהם שווים. בכדי לברר האם num2 הוא אכן הגדול יש לשאול שאלה נוספת.

נתרגם בעיה זו לתוכנית:

```
var str : string;
var num1 : number, num2: number;

str = prompt("Enter number1:", "");
num1 = parseInt(str);

str = prompt("Enter number2:", "");
num2 = parseInt(str);

if(num1 > num2)
{
    alert(num1);
}
else
{
    if(num1 < num2)
    {
        alert(num2);
    }
    else
    {
        alert("Equals");
    }
}
```

בחלק הראשון של התוכנית יש קליטה של שני משתנים.
לאחר הקליטה ישנה שאלה:
האם num1 גדול מ- num2?
אם כן, יודפס num1.
אם לא, נשאל שאלה נוספת: האם num1 קטן מ- num2?
אם כן, יודפס num2.
אם לא, יודפס "Equals".

ה- **else** השני שיין ל- if השני, ז"א שאם התוכנית לא תגיע לתנאי השני (במידה ו- num1 הוא הגדול) כמובן שהתוכנית גם לא תגיע ל- else השני.
אם התוכנית הגיעה ל- if השני, ז"א שבטוח ש- num1 אינו הגדול ולכן נותר לבדוק אם num1 הוא הקטן. אם לא, הברירה היחידה שנותרה היא שהם שווים.

אופרטור ! (Not)

האופרטור ! מאפשר להפוך תשובה של משפט בוליאני מ- true ל- false ולהיפך. האופרטור ! הופך את תוצאת התנאי של המשפט שכתוב אחריו בלבד, ז"א שאם ישנו משפט תנאי מורכב ונרצה להפוך את התוצאה הסופית של התנאי, עלינו להקיף את כל המשפט המורכב בסוגריים, ולא רק חלק ממנו.

לדוגמא:

```
var x : number = 10, y : number = 13;
var b : boolean;
b = !(x >= y);
alert('The result of '!(x >= y)' is ${b}');
```

פלט:

The result of '!(x >= y)' is true

כמובן שניתן לכתוב את המשפט גם כך (ללא שימוש באופרטור!):

```
b = x < y;
```

אך לעיתים כאשר ישנם משפטים מורכבים יותר נוח וברור לכתוב אותם על דרך השלילה.

שימוש נוסף אפשרי הוא כאשר נבדק כבר תנאי מסוים במהלך התוכנית, והתוצאה הוכנסה לתוך משתנה מטיפוס **boolean**, ובמקום אחר במהלך התוכנית נרצה לבדוק האם המשתנה הוא **false**, נוכל להיעזר באופרטור !.

```
if(!b)
{
    פקודות
}
```

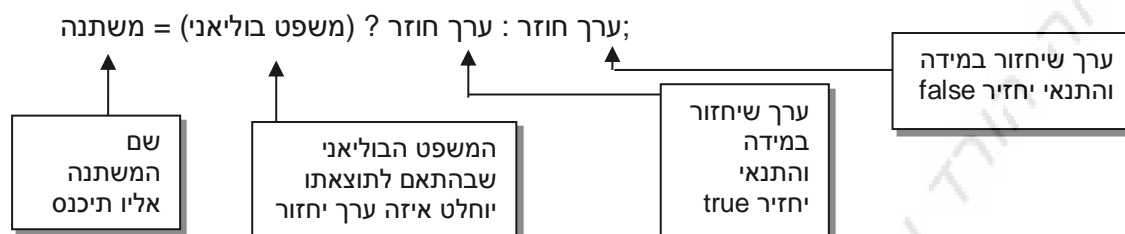
שזה כמו לכתוב:

```
if(b == false)
{
    פקודות
}
```

תנאי מקוצר

ישנם מקרים במהלך התוכנית בהם נרצה לבדוק תנאי, ובהתאם לתנאי להחליט איזה ערך יכנס למשתנה מסוים. נוכל לבצע זאת גם באמצעות תנאי מקוצר.

מבנה התנאי המקוצר:



בדוגמא הבאה יש בדיקה האם המשתנה x חיובי, כאשר אם כן תיכנס המחרוזת "Positive" לתוך המשתנה result ואם לא תיכנס המחרוזת "Negative" לתוך המשתנה result. לאחר מכן יש הדפסה של התוצאה.

```

var result : string;
var x : number = -5;
result = (x >= 0) ? "Positive" : "Negative";
alert(`${x} is result`);
  
```

פלט:

-5 is Negative

כמובן שניתן לכתוב זאת גם באמצעות תנאי if:

```

var result : string;
var x : number = -5;
if(x >= 0)
    result = "Positive";
else
    result = "Negative";

alert(`${x} is result`);
  
```

- תנאי מקוצר מיועד להחזרת ערך, ולכן לא ניתן לכתוב פקודה/פונקציה לביצוע (למעט פקודות המחזירות ערך כמו prompt).
- כל תנאי מקוצר ניתן להחליף בתנאי if (אך לא להיפך).
- בתנאי מקוצר האזורים האמורים להחזיר ערכים (במקרה של true/false) אינם בלוקים של קוד ולכן ניתן לכתוב פקודה אחת בלבד. זאת בניגוד לאזורים של if שניתן לכתוב בהם גם מספר פקודות.
- ניתן לכתוב בתנאי מקוצר גם משפטי תנאי מורכבים.
- לא נשתמש בתנאי מקוצר בכל מחיר. מומלץ להשתמש בתנאי מקוצר רק במידה והקוד ברור ולא מורכב מידי.

תנאי switch

לעיתים נרצה לבדוק ערך של משתנה, ועבור כל ערך לבצע פעולה אחרת. כמובן שנוכל לכתוב מספר תנאי if אחד אחרי השני, כאשר בכל תנאי ייבדק ערך אחר מול המשתנה. מנגנון התנאי switch מקל על הכתיבה.

מבנה תנאי switch:

(שם משתנה אותו נרצה לבדוק) switch

```
{
    case 1 אפשרות:
        פקודות שיבוצעו במידה וערך המשתנה הוא אפשרות 1
        break;
    case 2 אפשרות:
        פקודות שיבוצעו במידה וערך המשתנה הוא אפשרות 2
        break;
    case 3 אפשרות:
        פקודות שיבוצעו במידה וערך המשתנה הוא אפשרות 3
        break;
    .
    .
    .
    .
    default:
        פקודות שיבוצעו במידה והערך אינו מתאים לאף אחת מהאפשרויות
        break;
}
```

- בתוך הסוגריים שלאחר המילה **switch** נכתוב שם משתנה (לא תנאי בוליאני) כאשר המשתנה ייבדק מול כל אחד מה- case-ים.
- כל אפשרות case צריכה להסתיים ב- **break** כאשר ביניהם נכתוב את הפקודות המתאימות לאותה אפשרות.
- במידה ולא נכתוב **break**, בסוף של case מסוים והוא יהיה ה case המתאים למשתנה, יתבצע גם ה case שמופיע אחריו (אם מופיע).
- במידה ואף אחת מהאפשרויות (case-ים) לא מתאימה, יתבצעו הפקודות הנמצאות בבלוק ה- default.
- בלוק ה- default תמיד יופיע בסוף ולא ניתן לכתוב בו ערך שייבדק.
- במידה ולא נרצה לבצע פקודות כאשר אף אחת מהאפשרויות לא מתאימה, נוכל לוותר על בלוק ה- default (כולל ה- break שלו).

התוכנית הבאה קולטת מהמשתמש ערך, האמור להכיל את אחד מימות השבוע בפורמט מספרי (1 - 7). התוכנית מציגה את השם של היום (לדוגמא עבור 2 נדפיס Monday). במידה והמשתמש לא הכניס ערך נכון (קטן מ- 1 או גדול מ- 7), התוכנית תציג את הודעת השגיאה "Illegal day of week".

```
var dayOfWeek : number;
```

```
var str : string;
```

```
str = prompt("Enter day of week:", "");
```

```
dayOfWeek = parseInt(str);
```

```
switch(dayOfWeek)
```

```
{
```

```
    case 1:
```

```
        alert("Sunday");
```

```
        break;
```

```
    case 2:
```

```
        alert("Monday");
```

```
        break;
```

```
    case 3:
```

```
        alert("Tuesday");
```

```
        break;
```

```
    case 4:
```

```
        alert("Wednesday");
```

```
        break;
```

```
    case 5:
```

```
        alert("Thursday");
```

```
        break;
```

```
    case 6:
```

```
        alert("Friday");
```

```
        break;
```

```
    case 7:
```

```
        alert("Saturday");
```

```
        break;
```

```
    default:
```

```
        alert("Illegal day of week");
```

```
        break;
```

```
}
```

תיבת confirm

תיבת confirm הינה תיבה המאפשר לשאול את המשתמש שאלה, שהתשובה עליה היא בוליאנית. את התשובה שמשתמש ענה עלינו לקבל לתוך משתנה מטיפוס boolean. לדוגמא:

```
var b : boolean = confirm("Do you love java script?");
if(b)
    alert("Keep reading this book");
else
    alert("Go home");
```

בשורה הראשונה מוצגת תיבת ה - confirm, והתשובה שהמשתמש מזין נכנסת לתוך המשתנה b. מכיוון שהמשתנה b הוא בוליאני, ניתן לבדוק באמצעות תנאי if האם הוא true (המשתמש לחץ "אישור") או false (המשתמש לחץ "ביטול"), ובהתאם לתשובתו להציג לו הודעה מתאימה.

תיבת confirm:



תרגילים:



1. קלוט שני ערכים והצג "Growing..." אם השני גדול מהראשון.
2. קלוט שני ערכים והצג את הערך הגדול ביותר.
3. קלוט מספר שלם והצג "Even" אם הוא זוגי ו-"Odd" אם הוא אי-זוגי.
4. קלוט שני שלמים. הצג האם הראשון מתחלק בשני ובנוסף האם השני מתחלק בראשון.
5. קלוט שני ערכים והצג קודם את הערך הקטן ואת הערך הגדול אחריו.
6. קלוט מספר והצג: "חיובי", "שלילי" או "אפס" בהתאם לערכו.
7. קלוט מספר והצג אותו רק אם הוא גדול מ-928 או קטן מ-532.
8. בעל בית תוכנה החליט להעלות את המשכורת של כל תכניתן ב-10%, בתנאי שלאחר העלאה כזו הסכום לא יהיה גבוה מ-6,000 ש"ח. אם הסכום אכן יהיה גבוה מ-6,000 ש"ח, יקבל אותו תכניתן העלאה של 5% בלבד. קלוט את שמו של התכניתן ואת משכורתו הנוכחית. הצג את המשכורת של התכניתן לאחר ההעלאה.
9. קלוט שלושה מספרים. הצג "Increasing..." אם השני גדול מהראשון והשלישי גדול מהשני.
10. קלוט שלושה מספרים והצג את המספר הגדול ביותר.
11. קלוט מספר שלם בין 1 לבין 9,999. הצג את המספר ואת מספר ספרותיו.
12. במכבי תל אביב מחפשים שחקנים חדשים. תנאי הקבלה גיל בין 14 לבין 18 או בין 21 לבין 26. גובה השחקן חייב להיות מעל 182 סנטימטר. כתוב תוכנית הקולטת את הפרטים של שחקן אחד (גיל וגובה) ומדפיסה האם הוא התקבל או לא.
13. קלוט מספר בין 1 לבין 10 והצג את שמו (אחד, שתיים, ...). אם המספר חורג מהתחום הצג הודעת שגיאה.
14. קלוט שלושה מספרים והצג אותם מהקטן לגדול.
15. מנהל ביה"ס "יסוד" החליט שלא יופיעו בתעודות הציונים המספריים, אלא הערכה מילולית לפי המפתח הבא:
פחות מ-55 ← בלתי מספיק.
55 עד 64 ← מספיק.
65 עד 74 ← כמעט טוב.
75 עד 84 ← טוב.
85 עד 94 ← טוב מאד.
95 ומעלה ← מצוין.
קלוט ציון של תלמיד והצג את ההערכה המילולית המתאימה.
16. במערכת המשוואות הבאה A עד F הם מקדמים ו-x ו-y הם נעלמים:
$$A \times x + B \times y = C$$
$$D \times x + E \times y = F$$

ניתן לחשב את x ואת y ע"י נוסחאות העזר הבאות:
$$x = \frac{C \times E - B \times F}{A \times E - B \times D}$$
$$y = \frac{A \times F - C \times D}{A \times E - B \times D}$$

קלוט את המקדמים A עד F והצג את הפתרון.
המגע מחלוקה ב-0 בנוסחאות העזר!
אם המכנה 0, אין פתרון. במקרה כזה הצג "Equation has no solution".

פרק 6 – לולאות

בפרק זה נלמד את הנושאים הבאים:

- מהי לולאה
- לולאת תנאי
- לולאת תנאי מאוחר
- לולאת מונה
- לולאת תנאי מורכב
- לולאות מקוננות
- פקודות break ו-continue
- לולאה אינסופית
- לולאה ללא תנאי עצירה
- לולאה ריקה

מהי לולאה

לעיתים קרובות האלגוריתם הממומש דורש כי קטעי קוד מסוימים יבוצעו יותר מפעם אחת ויחידה. שימוש ב**לולאה** (loop) מאפשר לחזור מספר פעמים על קטע קוד. כל חזרה בודדת על קטע הקוד המופיע בלולאה נקרא איטרציה (**iteration**) של הלולאה.

נניח שנרצה לקלוט מהמשתמש 5 מספרים ולהציג לו את סכומם. כמובן שניתן לכתוב את פעולת הקלט 5 פעמים, ואם היינו רוצים 500 מספרים נכתוב את פעולת הקלט 500 פעמים? ואם מספר הקליטות לא ידוע בעת כתיבת התוכנית (לדוגמא: המשתמש יזין בתחילת התוכנית כמה מספרים הוא מתכוון להזין), האם נוכל לדעת מראש כמה קליטות לבצע?

במקרים אלו נשתמש בלולאה, אשר תפעיל קטע קוד מסוים מספר פעמים.

לולאת תנאי

לולאת תנאי - מאפשרת חזרה על קטע קוד עד שתנאי מסוים יתקיים או כל עוד תנאי מסוים מתקיים. מבנה לולאת התנאי while:

```
(משפט תנאי) while
{
    פקודות שיתבצעו כל עוד התנאי מתקיים
}
```

במידה ויש פקודה אחת בלבד בתוך הלולאה ניתן לוותר על הסוגריים התוחמים את הפקודות.

בדוגמא הבאה מוגדר משתנה בשם num ומאותחל לערך 0 מיד בשורת ההגדרה, וכן משתנה result המאותחל במחרוזת ריקה. לאחר מכן ישנה לולאת while אשר בתוכה שרשור של num למחרוזת result, ולאחר מכן שרשור של פסיק (,) למחרוזת. לאחר ההדפסה יש קידום של num ב-1. מכיוון שתנאי הלולאה הינו כל עוד num קטן מ-10, יודפסו כל המספרים מ-0 עד 9 ולאחר כל מספר יהיה פסיק (כולל פסיק לאחר ה-9). כאשר 0 הוא הערך של num לפני הכניסה ללולאה ו-9 הוא המספר האחרון המקיים את התנאי.

```
var num : number =0;
var result : string ="";
while(num < 10)
{
    result += num;
    result += ",";
    num++;
}
```

alert(result);

פלט:

0,1,2,3,4,5,6,7,8,9,

סדר הפעולות:

1. פקודות המופיעות לפני הלולאה.
2. התחלת הלולאה. בדיקת תנאי הלולאה:
 - במידה והתנאי מתקיים (true): כניסה לתוכן הלולאה (מעבר לשלב 3).
 - במידה והתנאי לא מתקיים (false): יציאה מהלולאה ללא ביצוע הפקודות שבתוכה (מעבר לשלב 4).
3. ביצוע הפקודות שבתוך הלולאה וחזרה לשלב 2.
4. פקודות המופיעות לאחר הלולאה.

בלולאת **while** יתכנו מקרים בהם הלולאה לא תתבצע אפילו פעם אחת, מכיוון שקודם כל מתבצעת בדיקת התנאי ורק לאחר מכן ביצוע פקודות, ואם התנאי לא מתקיים כבר בבדיקה הראשונה, הפקודות לא יתבצעו כלל. ניתן גם להבין מכך שמספר בדיקות התנאי בלולאת **while** גדול ב-1 ממספר הפעמים בהם יתבצעו הפקודות שבתוך הלולאה.

דוגמא נוספת:

נרצה לקלוט מספר לא שלילי (חיובי או אפס) מהמשתמש. במידה והמשתמש הקליד ערך שלילי עלינו לבקש ממנו מספר נוסף, ונניח שהמשתמש שלנו לא מהחכמים בעולם והוא מקליד שוב מספר שלילי, נצטרך לבקש ממנו מספר נוסף, ונוסף, ונוסף... עד אשר יהיה בידינו מספר לא שלילי.

כמה פעמים נצטרך לבקש מהמשתמש מספר נוסף?
אולי בניסיון הראשון הוא יקליד מספר תקין, ואולי רק בניסיון החמישי.
זהו המקום להשתמש בלולאת תנאי. נבקש מהמשתמש מספר ונמשיך לבקש כל עוד המספר שהוקלד שלילי.

```
var num : number;

var tmp : string = prompt("Enter a non negative number:","");
num = parseInt(tmp);

while(num < 0)
{
    tmp = prompt("Enter a non negative number:","");
    num = parseInt(tmp);
}
```

דוגמא נוספת:

קלוט מספר שהינו תוצאה של כפולות של 2, והצג מהן מספר הכפולות, כלומר מהי החזקה.

```
var number, powerCount=0;

number = prompt("Enter a number:","");
number = parseInt(number);

while(number >= 2)
{
    number /= 2;
    powerCount++;
}

alert(powerCount);
```

נעקוב אחר הקוד:

הגדרת המשתנים:

number - לתוכן נקלוט את המספר.

powerCount - משתנה זה יהווה מונה למספר הכפולות של 2.

נקלוט מספר לתוך המשתנה number.

התחלת הלולאה - הלולאה תתבצע כל עוד number גדול או שווה ל- 2.

אם נכנסנו ללולאה נחלק את number בשתיים ויש להעלות את powerCount ב- 1.

נציג את התוצאה של powerCount.

נרחיב את ההסבר:

נקטין את המספר שקלטנו פי 2 בכל פעם שנבצע את הלולאה עד שיגיע ל- 1.

לדוגמא: נניח וקלטנו את המספר 16, אזי בפעם הראשונה של הלולאה:

```
powerCount = 0 + 1
number = 16 / 2
```

בשלב הזה $number = 8$, והלולאה תמשיך:

```
powerCount = 1 + 1
number = 8 / 2
```

וכן הלאה:

```
powerCount = 2 + 1
number = 4 / 2
```

והלאה:

```
powerCount = 3 + 1
number = 2 / 2
```

בשלב הזה $number = 1$ והלולאה תיעצר.
אם כך ביצענו את הלולאה 4 פעמים ו- $powerCount$ יכול את הערך 4. ז"א ש- 2 בחזקת 4 הינו 16.

לולאת תנאי מאוחר

לעיתים נדרש "להריץ לולאת תנאי" כאשר ידוע מראש שעל קטע הקוד המופיע בלולאה, להתבצע לפחות פעם אחת. ואז, כל עוד התנאי מתקיים הפעולה תחזור על עצמה פעם שנייה, שלישית, וכך הלאה.

מבנה לולאת התנאי **do...while**:

```
do
{
    פקודות שיתבצעו כל עוד התנאי מתקיים
}while(משפט תנאי);
```

במידה ויש פקודה אחת בלבד בתוך הלולאה ניתן לוותר על הסוגריים התוחמים את הפקודות.

סדר הפעולות בלולאת **do...while**:

1. פקודות המופיעות לפני הלולאה.
2. ביצוע הפקודות שבתוך הלולאה.
3. בדיקת התנאי:
- במידה והתנאי מתקיים (true): חזרה לשלב 2.
- במידה והתנאי לא מתקיים (false): מעבר לשלב 4.
4. פקודות המופיעות לאחר הלולאה.

אם נסתכל כך על סדר הפעולות נראה שסדר הפעולות הינו בדיוק כמו בלולאת while, למעט שלב 2. מכאן אנו למדים שבניגוד ללולאת while, בלולאת do...while אנו יכולים להיות בטוחים שהפקודות שבתוך הלולאה יתבצעו לפחות פעם אחת, אפילו אם התנאי לא מתקיים מיד בכניסה ללולאה. ניתן גם להבין מכך שמספר בדיקות התנאי בלולאת do...while שווה למספר הפעמים בהם יתבצעו הפקודות שבתוך הלולאה (בניגוד ללולאת while).

ניקח את הדוגמא שראינו בלולאת while, ונהפוך אותה ללולאת do...while.

הדוגמא הבאה אמורה לקלוט מהמשתמש מספר עד שיקליד מספר לא שלילי. ניתן לראות שהקוד המופיע לפני הלולאה מתבצע (קליטה), ובתוך הלולאה מתבצע בדיוק אותו הקוד. זאת מכיוון שנרצה קודם לקלוט מספר מהמשתמש, ורק אם הוא שלילי נרצה להיכנס ללולאה ולקלוט שוב עד שיקליד מספר לא שלילי. זהו מקרה קלאסי בו נרצה להשתמש בלולאת do...while בכדי לא לכתוב את אותו הקוד פעמיים, ובכל זאת לבצע אותו לפחות פעם אחת.

לולאת while:

```
var num : number ;

var tmp : string = prompt("Enter a non negative number:","");
num = parseInt(tmp);

while(num < 0)
{
    num = prompt("Enter a non negative number:","");
    num = parseInt(num);
}
```

לולאת do...while:

```
var num : number;

do
{
    let tmp : string = prompt("Enter a non negative number:","");
    num = parseInt(tmp);
}while(num < 0);
```

לולאת מונה

לולאת מונה - מאפשרת חזרה על קטע קוד כל עוד תנאי מסוים יתקיים, כאשר ניתן להוסיף לה מונה מובנה. מונה הלולאה הינו משתנה אשר עובר שינוי בכל איטרציה של הלולאה. לולאה זו הינה הלולאה הנפוצה ביותר. כפי שניתן לראות מההגדרה הנ"ל, לולאת מונה הינה למעשה לולאת תנאי עם תוספת של מונה מובנה. מכיוון שמאד נפוץ להשתמש בלולאת תנאי, אשר מכילה מונה, נעדיף במקרה זה להשתמש בלולאת מונה.

מבנה לולאת מונה for:

```
for(קידום המונה ; משפט תנאי ; אתחול המונה)
{
    פקודות שיתבצעו כל עוד התנאי מתקיים
}
```

במידה ויש פקודה אחת בלבד בתוך הלולאה ניתן לוותר על הסוגריים התוחמים את הפקודות. ניזכר בדוגמא הראשונה אותה ראינו בלולאת while, דוגמא זו מדפיסה את המספרים מ-0 עד 9.

```
var num : number =0;
var result : string ="";
while(num < 10)
{
    result += num;
    result += ",";
    num++;
}
```

alert(result);

פלט:

0,1,2,3,4,5,6,7,8,9,

1. כפי שניתן לראות, יש בלולאה זו את 3 האלמנטים הבאים:
 1. אתחול המונה num ל- 0 (לפני הלולאה).
 2. משפט תנאי העוצר את הלולאה כאשר num מגיע ל- 10.
 3. קידום המונה num בתוך הלולאה.

נהפוך את הקוד שבדוגמא ללולאת for, כך שיבצע את אותו הדבר:

```
var num : number ;
var result : string ="";

for(num=0 ; num < 10 ; num++)
{
    result += num;
    result += ",";
}
```

alert(result);

גם כאן כמובן הפלט הוא:

0,1,2,3,4,5,6,7,8,9,

ניתן לראות שכל האלמנטים הנ"ל מובנים בצורה נוחה יותר בתוך הסוגריים שבלולאת for. סדר הפעולות בלולאת for:

1. פקודות המופיעות לפני הלולאה.
2. פקודת אתחול המונה.
3. בדיקת התנאי:
- במידה והתנאי מתקיים (true): מעבר לשלב 4.
- במידה והתנאי לא מתקיים (false): מעבר לשלב 6.
4. ביצוע הפקודות שבתוך הלולאה.
5. פקודת קידום המונה וחזרה לשלב 3.
6. פקודות המופיעות לאחר הלולאה.

כמו שניתן לראות סדר הפעולות דומה מאוד ללולאת while, כאשר ההבדלים הם שלפני הכניסה ללולאה מופעלת פקודת האתחול, ובכל סיום של איטרציה מופעלת פקודת קידום המונה.

ניתן להגדיר את המונה של הלולאה בתוך הסוגריים של הלולאה. חשוב להבין שלמרות שהמשתנה הוגדר בתוך הלולאה, ניתן יהיה להשתמש בו לאחר הלולאה (בניגוד לשפות אחרות שלא מאפשרות זאת).

הגדרת משתנה המונה בתוך הלולאה:

```
var result : string = "";

for(let num : number =0 ; num < 10 ; num++)
{
    result += num;
    result += ",";
}

alert(result);
```

כמו שנאמר קודם, לולאת for מאפשרת לאתחל משתנה מונה ולקדם אותו באופן מובנה בתחביר הלולאה. ניתן להוסיף מספר בלתי מוגבל של קידומים ואיתחולים באותה הלולאה.

הדוגמא שלהלן מדפיסה את העצרת של המספרים מ-1 עד 10:
(עצרת של מספר היא מכפלת כל המספרים מ-1 ועד לאותו מספר לדוגמא: $4! = 1*2*3*4$)

```
var result : string = "";

for (let i = 1, let j = 1 ; i <= 10; i++ , j*=i)
{
    result += i + "!" + " = " + j + "\n";
}

alert(result);
```

פלט:

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

ניתן לראות שבלולאה הוגדרו שני מונים (i,j) כאשר המונה i מונה את המספרים עליהם נרצה לחשב את העצרת, והמונה j מכיל את העצרת עצמה. המונה i יקודם בכל איטרציה של הלולאה ב-1, והמונה j יכפיל את עצמו ב-i, כך שבכל איטרציה של הלולאה j יכיל את העצרת של i. בדוגמא זו כל החישובים נכתבו באופן מובנה בתוך התחביר של הלולאה.

ניתן גם לוותר על אחד או יותר מהחלקים המובנים בלולאה קידום/אתחול/תנאי.

בדוגמא הבאה נרצה לקלוט מספר מהמשתמש ולהדפיס את כל המספרים מהמספר שנקלט ועד למספר 10. מכיוון שאין אנו יודעים מראש איזה מספר המשתמש יכניס אנו לא יכולים לאתחל את המונה ובמקרה זה נוותר על חלק האתחול.

```
var result : string = "";
var x : number = parseInt(prompt("Enter number: ", ""));

for ( ; x <= 10; x++)
{
    result += x + ", ";
}

alert(result);
```

פלט עבור קלט 3:

3,4,5,6,7,8,9,10,

שאלה: מה יקרה אם המשתמש יכניס מספר גדול מ-10 ?
תשובה: הלולאה לא תתבצע כלל מכיוון שהתנאי נבדק בפעם הראשונה לפני הכניסה ללולאה, ומכיוון שהתנאי לא יהיה נכון כבר בפעם הראשונה הלולאה לא תתבצע כלל.

לולאת תנאי מורכב

החוקים לגבי משפטי התנאי שניתן להגדיר בתוך לולאה הינם בדיוק כמו החוקים של משפטי התנאי שניתן להגדיר בתוך if. כולל סוגי האופרטורים לבדיקה ותנאים מורכבים. דוגמא ללולאת תנאי מורכב:

```
var num : number = -1, sum : number = 0;
alert("Enter 10 numbers. To stop enter 0.");

for (let i = 1; i <= 10 && num != 0; i++)
{
    num = parseInt(prompt("Enter number " + i, ""));
    sum += num;
}

alert(`Sum = ${sum}`);
```

הדוגמא הנ"ל קולטת מהמשתמש 10 מספרים, מחשבת את סכומם ומדפיסה את הסכום בסוף התוכנית. בתוכנית זו נרצה לאפשר למשתמש לעצור גם אחרי פחות מ-10 מספרים אם ירצה, כך

שכאשר יכניס את המספר 0 נעצור את הקליטה ונדפיס את הסכום. לשם כך נצטרך תנאי מורכב אשר יעצור את הלולאה לאחר 10 מספרים וגם יעצור את הלולאה במידה והמשתמש הקליד 0.

לולאות מקוננות

ישנם בתכנות הרבה מקרים בהם נרצה לכתוב לולאה ועל כל הלולאה נרצה לחזור מספר פעמים. לשם כך נשתמש בלולאות מקוננות.

לולאה מקוננת - לולאה הכתובה בתוך לולאה אחרת.

לדוגמא:

אדם בעל אמצעים החליט לבנות בית קולנוע פרטי. האולם כולל 4 שורות של כורסאות. בכל שורה 5 כורסאות. החליט האדם למספר שורות וכורסאות לפי המספור הבא:

<u>1</u>	1	2	3	4	5
<u>2</u>	1	2	3	4	5
<u>3</u>	1	2	3	4	5
<u>4</u>	1	2	3	4	5

נרצה להציג מספור זה על המסך. לצורך כך נדרש לארבע לולאות for. אילו אדם זה החליט לבנות בית קולנוע בעל 100 שורות של כורסאות. ידרשו 100 לולאות for להצגת מספור הכורסאות באולם החדש. זה לא נבון.

לפתרון נבון נוצר מנגנון קינון הלולאות.

```
var result="";

for (var i = 1; i <= 4; i++)
{
    result += i + " ";

    for (var j = 1; j <= 5; j++)
    {
        result += j + " ";
    }

    result += "\n";
}

alert(result);
```

פלט:

```
1: 1 2 3 4 5
2: 1 2 3 4 5
3: 1 2 3 4 5
4: 1 2 3 4 5
```

הלולאה הפנימית מדפיסה את המספרים מ-1 עד 5. מכיוון שלולאה זו צריכה להתבצע 4 פעמים, "נעטוף" לולאה זו בלולאה (החיצונית) אשר תתבצע 4 פעמים. לפני הלולאה הפנימית נדפיס את מספר השורה בתוספת נקודתיים (:). ולאחר הלולאה הפנימית נבצע ירידת שורה.

פקודות break ו-continue

בשפת Typescript ניתן לעצור לולאה "באופן יזום" גם כאשר תנאי הסיום עדיין לא התקיים. מומלץ לתכנן לולאות אשר יסתיימו בתנאי הסיום, אך לעיתים בלולאות מורכבות רצוי לפשט את הלולאה ולשבור אותה באמצע, כדי להקל על הבנת הקוד.

ניתן גם להפסיק איטרציה מסוימת של הלולאה ולעבור לאיטרציה הבאה. גם פעולה זו לא מומלצת לביצוע שלא לצורך, אך לעיתים כדאי לבחור בפעולה זו כדי להקל על הבנת הקוד.

break - פקודה אשר שוברת/עוצרת את הלולאה. לאחר פקודה זו התוכנית תעבור לקוד הכתוב אחרי הלולאה.

continue - פקודה אשר מפסיקה את האיטרציה הנוכחית של הלולאה. לאחר פקודה זו התוכנית תעבור לתחילת הלולאה (בדיקת התנאי) ותמשיך לאיטרציה הבאה.

דוגמא לפקודת break:

```
var num : number = parseInt(prompt("Enter number: ", ""));
```

```
var tmp : number;
var i : number;
for (i = 2; i < num; i++)
{
    tmp = num % i;
    if (tmp == 0)
        break;
}
```

```
if(i == num)
    alert("primary");
else
    alert("not primary");
```

בדוגמא הנ"ל נקלט מספר מהמשתמש, ונדפיס האם המספר הוא ראשוני (מספר המתחלק רק בעצמו וב-1 ללא שארית). בכדי לבדוק זאת יש לבדוק האם המספר מתחלק ללא שארית עבור כל אחד מהמספרים מ-2 ועד לאותו מספר (לא כולל המספר עצמו). במידה והוא מתחלק במספר מסוים, אין צורך להמשיך ולבדוק את כל המספרים הבאים וניתן לעצור את הלולאה (לדוגמא: אם המספר שנקלט הוא 8 ניתן לעצור כבר בבדיקה הראשונה כי 8 מתחלק ב-2 ללא שארית). לאחר הלולאה ניתן לבדוק האם מונה הלולאה הגיע עד למספר, אם כן זה אומר שהמספר שנקלט לא התחלק (ללא שארית) לאף מספר ולכן הוא ראשוני ואם לא זה אומר שהלולאה נעצרה באמצע והמספר אינו ראשוני.

דוגמא לפקודת continue:

```
var num : number = 0, sum : number = 0;
alert("Enter 10 number:");

for (let i = 1; i <= 10; i++)
{
    num = parseInt (prompt("Enter number " + i, ""));

    if(num < 0)
        continue;

    sum += num;
}

alert("Sum = " + sum);
```

בדוגמא הנ"ל נקלוט מהמשתמש 10 מספרים, ונסכום רק את המספרים הלא שליליים. כאשר המשתמש יקליד מספר שלילי לא נבצע את פעולת החיבור, ונעבור לקליטת המספר הבא (האיטרציה הבאה של הלולאה).

דברים נוספים לגבי הפקודות break ו-continue:

- כל פעולה שניתן לבצע בעזרת פקודות אלו, ניתן לבצע גם בלעדיהם.
- פקודות אלו ניתן לכתוב בכל אחת מסוגי הלולאות.
- פקודות אלו חייבות להיות בתוך תנאי מסוים אחרת הן תמיד יתבצעו, ואין לכך משמעות.
- פקודות אלו חוקיות רק בתוך לולאות (למעט פקודת break שמשמשת גם בתנאי switch).

לולאה אינסופית

לולאה אשר לא תיעצר לעולם נקראת לולאה אינסופית.

אם נכתוב לולאה עם תנאי סיום, אשר תנאי זה לא יתקיים לעולם, הלולאה תמשיך ולא תיעצר (אלא אם נסגור את התוכנית "בכוח"). יש לשים לב שלא להיקלע למקרה כזה כדי שהתוכנית לא תתקע.

דוגמא ללולאה אינסופית:

```
var x : number = 0, num : number = 0;

while (x < 10)
{
    num++;
}

alert(`num = ${num}`);
```

בדוגמא זו תנאי העצירה ($x < 10$) לעולם לא יקרה, שכן x מאותחל לערך 0 לפני הלולאה ולא משתנה בתוך הלולאה, כך שכל הזמן הוא יישאר קטן מ-10. כמובן שהתוכנית לעולם לא תגיע לשורה שאחרי הלולאה.

לולאה ללא תנאי עצירה

ניתן לכתוב לולאה ללא תנאי עצירה. שימוש בלולאה כזו אינו נפוץ, אך לעיתים בכדי להקל על כתיבת והבנת הקוד נשתמש בסוג כזה של לולאה. בכדי לצאת מלולאה אשר אין בה תנאי עצירה, נשתמש בפקודת break.

לולאת while ללא תנאי עצירה:

```
while (true)
{
    ...
}
```

לולאת do...while ללא תנאי עצירה:

```
do
{
    ...
} while (true);
```

לולאת for ללא תנאי עצירה:

```
for (i = 0 ; ; i++)
{
    ...
}
```

לולאה ריקה

לולאה ריקה הינה לולאה אשר לא מכילה פקודות.

לעיתים נרצה לבצע חישוב מסוים, שניתן לבצע אותו כבר בסוגריים של הלולאה (היכן שנמצא התנאי) ואין לנו צורך לכתוב פקודות בתוך הלולאה. במקרה זה נשתמש בלולאה ריקה.

לולאה ריקה נכתבת רק עם התנאי ובתוספת נקודה-פסיק (;) לאחר הסוגריים של הלולאה.

מבנה לולאה ריקה:

while (תנאי) ;

for (קידומים ; תנאי ; איתחולים) ;

אין צורך לכתוב לולאת do..while ריקה שכן היא תתבצע בדיוק כמו לולאת while ריקה.

לדוגמא חישוב עצרת של מספר:

```
var num : number , res : number =1;
num = parseInt (prompt("Enter number:", ""));
for (var i = 1; i < num; i++, res *= i) ;

alert(num + "! = " + res);
```

בדוגמא הנ"ל המשתמש מכניס מספר והתוכנית מציגה את העצרת של המספר. חישוב העצרת יכול להיעשות כבר בסוגריים של הלולאה (היכן שנמצא התנאי) ואין צורך לכתוב פקודה נוספת בתוך הלולאה.



תרגילים:

בכל התרגילים הבאים יש לעבוד עם מספרים שלמים.

1. קלוט מספר חיובי top. הצג את כל המספרים הטבעיים מ-1 עד top (כולל).
2. קלוט שני מספרים והצג את כל המספרים ביניהם בסדר עולה (כולל המספרים עצמם). לא מובטח שהנתון השני גדול מהראשון.
3. קלוט מספר n. הצג את כל המספרים הזוגיים מ-0 עד n. לא מובטח ש-n זוגי.
4. קלוט שני מספרים max ו-n. הצג את כל המספרים השלמים עד max (כולל) המתחלקים ב-n. לא מובטח ש-max עצמו מתחלק ב-n.

עיבוד נתונים...

5. בתור הקלט כמות לא ידועה של מספרים לא שליליים שבסופם המספר 99. הצג את סכום המספרים לא כולל ה-99 (יתכן והנתון הראשון הוא 99).
6. בתור הקלט כמות לא ידועה של מספרים חיוביים שבסופם המספר 0. הצג את ממוצע כל המספרים לא כולל ה-0 (יתכן והנתון הראשון הוא 0).
7. קלוט מספרים עד שתקלוט מספר שלילי או 0. הצג את הערך הגבוה ביותר שנקלט (יתכן והנתון הראשון אינו חיובי).
8. קלוט מספרים עד שתקלוט מספר שלילי או 0. הצג את הערך החיובי הנמוך ביותר (יתכן והנתון הראשון אינו חיובי).
9. קלוט 10 מספרים. הצג את המספר הסידורי של הערך הגבוה ביותר.

ניתוח שלם...

10. קלוט מספר והצג את הספרה השמאלית ביותר שלו.
11. קלוט מספר והצג את מספר הספרות שלו.
12. קלוט מספר והצג את סכום הספרות שלו.
13. קלוט מספר X ושלם חד ספרתי dig. הצג כמה פעמים הספרה dig מופיעה ב-X.
14. קלוט מספר והצג את ספרותיו בסדר הפוך (עבור הקלט 1304 יוצג 4031).
15. פולינדרום הוא מספר סימטרי, כלומר ערכו זהה גם כשרושמים את ספרותיו בסדר הפוך (למשל: 12321, 4774). קלוט מספר שלם, והצג האם הוא פולינדרום.

חשבון...

16. קלוט שני מספרים לא שליליים. הצג את מכפלתם ללא שימוש באופרטור הכפל.
17. קלוט שני מספרים חיוביים. הצג את הראשון בחזקת השני ללא שימוש באופרטור החזקה.

18. קלוט שני מספרים לא שליליים. הצג את המנה השלמה ואת השארית ללא שימוש באופרטור החילוק או המודולו.
19. עצרת של מספר n היא מכפלת כל השלמים בין 1 לבין n (כולל).
למשל: העצרת של 5 היא $120 = 5 \times 4 \times 3 \times 2 \times 1$.
העצרת של 2 היא $2 = 2 \times 1$.
קלוט מספר טבעי n והצג את העצרת שלו.
20. קלוט מספר והצג את השורש הריבועי שלו בדיוק של 1% ללא שימוש באופרטור החזקה. דיוק של 1% - אם מחפשים לדוגמא את השורש של 1000, מספיק למצוא שורש של מספר בין 990 לבין 1010.

סדרת פיבונצ'י...

21. סדרת פיבונצ'י מוגדרת באופן הבא:
האיבר הראשון שווה 1.
האיבר השני שווה האיבר הראשון גם.
כל איבר נוסף שוו לסכום שני האיברים שלפניו.
כלומר, הסדרה מתחילה כך: $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \dots$
א. קלוט מספר $index$ והצג את האיבר ה- $index$ בסדרה. (מובטח כי $index > 2$)
ב. קלוט מספר val . הצג את סידרת פיבונצ'י עד האיבר הראשון הגדול מ- val .

מתמטיקה...

22. קלוט מספר והצג את כל המחלקים השלמים שלו.
למשל: המחלקים של 60 הם $1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60$.
המחלקים של 61 הם $1, 61$.
23. קלוט שני מספרים והצג את המחלק המשותף הגדול ביותר.
למשל: המחלק הגדול ביותר של 60 ושל 72 הוא: 12.
המחלק הגדול ביותר של 64 ושל 81 הוא: 1.
24. קלוט מספר והצג "ראשוני" או "לא ראשוני" בהתאם. (מספר ראשוני הוא מספר המתחלק בשלמות רק בעצמו וב-1).
25. קלוט מספר והצג את פירוקו לגורמים ראשוניים.
למשל: 60 מתפרק ל- $2 \times 2 \times 3 \times 5$.
59 מתפרק ל- 1×59 (59 עצמו הוא ראשוני).

מספר לולאות...

26. קלוט שני מספרים. הצג את כל המספרים שביניהם מהראשון שנקלט ועד השני שנקלט.
27. קלוט שני מספרים. הצג את כל המספרים שביניהם מהקטן לגדול ובחזרה לקטן.
28. בתור הקלט רשימת טמפרטורות היומיות של יולי (30 ימים) 2000 ו-2001 (סה"כ 60 מספרים).
חשב את הטמפרטורה הממוצעת ביולי 2000.
הצג את הימים ביולי 2001 בהם החום היה גבוה מהממוצע של יולי 2000.
29. קלוט מספרים עד שייקלט ערך שלילי. עבור כל שלם חיובי הצג את סכום ספרותיו.
30. קלוט מספר והצג את ספרותיו ממוינות בסדר עולה.
למשל: קלט – 5001, פלט – 0015.
קלט – 48444, פלט – 44448.

31. קלוט מספרים עד שייקלט מספר ראשוני. הצג את כמות המספרים השלמים הלא-ראשוניים שנקלטו.
32. קלוט מספרים עד שייקלט 0. עבור כל מספר הצג האם הוא איבר בסדרת פיבונצ'י:
 $\rightarrow 1, 1, 2, 3, 5, 8, 13, 21, \dots$
33. קלוט מספר לתא N. חשב והדפס את N עצרת לפי הנוסחה: $S! = 1 * 2 * 3 * \dots * N$
34. קלוט מספר לתא N. חשב והדפס את סכום המספרים מ-1 ועד N אשר מתחלקים ב-3 (למשל: אם הנתון היה 7, התוצאה תהיה 9 כי עד 7 המספרים 3 ו-6 מתחלקים ב-3).
35. קלוט 15 מספרים, סכם כל מספר שלישי (כלומר, לסכם את הנתון ה-3, 6, 9,). והדפס את התוצאה.
36. קלוט מספר לתא N. סכם את כל המספרים עד N המתחלקים ב-4 ואת כל המתחלקים ב-7, לסוכם אחד. הדפס את התוצאה. (כמה לולאות צריך לפתרון השאלה?)
37. קלוט 20 מספרים. הצג את סכום כל הזוגיים.
38. קלוט 20 מספרים. הצג את סכום כל הנתונים שמספרם הסיידורי זוגי (ה-2, 4, 6 וכו'...).
39. קלוט מספר לתא A. לאחר מכן קרא 20 מספרים והדפס את מספרם הסיידורי של אלה השווים ל-A.
40. קלוט מספר N, ועוד N מספרים. הצג את הערך הגדול ביותר מבין המספרים, המספר הסיידורי של המופע הראשון של אותו ערך ואת מספר המופעים הנוספים של אותו הערך.
41. קלוט מספר N, ועוד N מספים. הצג את הערך השני בגודלו מבין המספרים ואת המספר הסיידורי של המופע האחרון של אותו ערך בקלט.
42. קלוט 10 מספרים ובדוק אם הם ממוינים בסדר עולה (כלומר אם כל נתון הוא גדול או שווה לקודמו). אם כן הדפס: "ממוין" אחרת הפסק והדפס: "לא ממוין".
43. קלוט 20 מספרים וסכם אותם, כאשר הגעת לנתון השווה לסכום כל קודמיו הפסק והדפס אותו. אם לא מצאת כזה הדפס "לא נמצא".
44. בבי"ס מסוים יש 5 תלמידים ולכל תלמיד 5 ציונים. הנתונים מסודרים בקלט לפי תלמידים דהיינו - 5 ציוני תלמיד ראשון, אח"כ 5 ציוני תלמיד שני וכו'. הדפס את ממוצע ציוניו של כל תלמיד וכן את ממוצע בי"ס. (ממוצע ביה"ס הנו ממוצע כל הציונים שנקלטו).
45. קלוט מספר N. והדפס את הדבר הבא. בהנחה ש- $N = 4$:
- 1 2 3 4
2 3 4
3 4
4
46. כמו בשאלה הקודמת רק שהפעם תודפס סדרה כזו:
- 4 3 2 1
4 3 2
4 3
4
47. קלוט 10 מספרים והצג את כל המספרים בין כל זוג מספרים שנקלטו.
 למשל: קלט – 9, 12, 8, 8...
 פלט – 9, 10, 11, 12, 11, 10, 9, 8, 8...

48. קלוט מספר שורות ומספר עמודות והצג מלבן כוכביות. עבור הקלט 5 ו-3:

* * *
 * * *
 * * *
 * * *
 * * *

49. הדפס את לוח הכפל.

50. הדפס את כל המספרים בין 10 ל-99, שספרת האחדות שלהם גדולה מספרת העשרות.

51. קלוט ספרה, הדפס את כל המספרים מ-1 ועד 100 שהספרה מופיעה בהם.

52. הדפס את כל המספרים הדו-ספרתיים המתחלקים לספרת האחדות שלהם, לספרת העשרות שלהם, ולסכום הספרות שלהם.

53. נתונים ערכי השטחות הבאים: 200 ש"ח, 100 ש"ח, 50 ש"ח, 20 ש"ח, וערכי המטבעות הבאים:

10 ש"ח, 5 ש"ח, 1 ש"ח, 0.5 ש"ח, 10 אגורות, 5 אגורות, 1 אגורה.

קלוט סכום לתשלום, הדפס את הסכום בעזרת מינימום שטרות ומטבעות.

54. הדפס את כל המספרים התלת ספרתיים שכל ספרותיהם זוגיות.

פרק 7 – פונקציות

בפרק זה נלמד את הנושאים הבאים:

- הסבר כללי
- מבנה בסיסי של פונקציה
- קבלת פרמטרים לפונקציה
- החזרת ערך מפונקציה
- שמות פונקציות
- טווח הכרה של משתנים
- שימוש בפונקציות קיימות
- סיכום

הסבר כללי

אנשי פיתוח "לימדו" את המחשב לבצע פעולות מסוימות, ובאופן טבעי אנו מפעילים אותם ללא הרף מבלי להמציאם מחדש.

נחשוב על מקש ה-Enter, בכל לחיצה הוא מבצע את פעולתו נאמנה, לא צריך להמציא אותו, המציא אותו עבורנו. נחשוב על המנגנון alert() המיועד להצגת מידע על המסך. "לימדו" או במילים אחרות תכנתו את המחשב להציג מידע על המסך. לשמחתנו מנגנון זה פועל ואנו מפעילים אותו בעת הצורך. מוצר תוכנה מורכב ממנגנונים רבים. אלמלא מנגנונים אלה היינו צריכים להמציא אותם כל פעם מחדש, תהליך לא יעיל הגוזל זמן רב.

במוצרי התוכנה שנפתח נשתמש במנגנונים קיימים כדי "לא להמציא את הגלגל מחדש" וכן במנגנונים שאנו נפתח. מנגנונים אלה מכונים פונקציות.

ככל שהתוכניות הולכות ונהיות גדולות ומורכבות יותר הקוד הופך ארוך ומסורבל, עד אשר אינו ניתן לשליטה והבנה. הפתרון לבעיה זו הנו לחלק את התוכניות לחלקים. במילים אחרות, במקום לצקת את כל הבניין בבת אחת ניצוק קומה, נודא שהיא יציבה ואז נמשיך לקומה הבאה. **פונקציה** היא אבן הבניין של תוכניות. כל פונקציה היא בעצם מיני תוכנית, הכוללת משתנים משל עצמה, המבצעת פעולה סגורה ומוגדרת. בכל תוכנית נמצא פונקציות.

מנגנון הפעולה הוא כזה: התוכנית "קוראת" לפונקציה. הפונקציה מבצעת את פעולתה ומחזירה את השליטה לתוכנית. פונקציה יכולה גם לקרוא לפונקציות אחרות. לכשהן יסיימו את פעולתן השליטה תחזור אליה.

קטע קוד לביצוע משימה מוגדרת העונה לשם כלשהו ומופעל על ידי "קריאת שמו" הנו פונקציה. לעיתים נאלץ לשכפל קטעי קוד לביצוע משימה דומה, כמו חישוב עצרת במקומות שונים בתוכנית. שיכפול זה **אסור!** מכיוון שהקוד מתארך, מורכבותו גדלה והשליטה בקוד קשה יותר. התחזוקה (תחזוקה היא תיקון תקלות, שיפור הקיים והוספת תכונות חדשות במרוצת הזמן) הופכת בשלבים מסוימים לבלתי ניתנת לשליטה עד כדי כתיבת התוכנית מחדש. לכן סביבת הדפדפן מכילה פונקציות רבות העומדות לשירותנו בכל עת. בנוסף מאפשרת לנו הסביבה ליצור פונקציות משלנו ככל שידרש. באופן זה הקוד שאנו כותבים "נשלט" טוב יותר, אין צורך "לראות" עשרות ומאות אלפי שורות קוד כדי לתקן תקלה, או לשפר תפקוד. פשוט נגיע לפונקציה הנוגעת לשינוי המבוקש.

יתרונות לשימוש בפונקציות:

- ◆ שימוש חוזר בקטעי קוד, ומניעת שיכפול של קוד.
- ◆ חלוקה לוגית של התוכנית לקטעים "נשלטים".
- ◆ תחזוקה נוחה יותר.
- ◆ המשתנים המוגדרים בפונקציה מוכרים רק לה!

מבנה בסיסי של פונקציה

להלן המבנה הבסיסי של פונקציה ב - Typescript:

```
function funcName() : void
{
    תוכן הפונקציה:
    כאן נכתוב את הפקודות שיתבצעו כאשר נפעיל את הפונקציה
}
```

להלן דוגמא אותה ננתח:

```
printName();
```

```
function printName() : void
{
    alert("My name is 'Avi'");
}
```

הפונקציה **printName** מציגה את הטקסט: "My name is 'Avi'"
בכדי להפעיל את הפונקציה שבנינו עלינו לקרוא בשמה בתוספת סוגריים:

```
printName();
```

פלט התוכנית:

My name is 'Avi'

ניתן להפעיל את אותה הפונקציה גם מספר פעמים:

```
printName();
printName();
printName();
```

```
function printName() : void
{
    alert("My name is 'Avi'");
}
```

פלט התוכנית:

My name is 'Avi'
My name is 'Avi'
My name is 'Avi'

כבר בשלב זה ניתן להבחין ביתרון שבשימוש בפונקציה, אין צורך לכתוב את אותו הקוד מספר פעמים. ניתן לבנות פונקציה ולקרוא לה כמה פעמים שנרצה. ניתן לכתוב בתוך פונקציה כמה שורות קוד שנרצה, ובכל פעם שנפעיל אותה יתבצעו כל שורות הקוד.

פונקציה עם מספר שורות קוד:

```
readAndPrintName();
```

```
function readAndPrintName() : void
{
    var name : string = prompt("Enter your name: ", "");
    alert(`Your name is: ${ name }`);
}
```

הפונקציה **readAndPrintName** קולטת מהמשתמש את שמו ומדפיסה את הערך שנקלט.

כאשר נרצה להפעיל / לקרוא לפונקציה מסוימת, נוכל להפעיל אותה גם מתוך פונקציה אחרת.

הפעלת פונקציה מפונקציה אחרת:

```
readAndPrintNameAndAge();
```

```
function readAndPrintNameAndAge() : void
{
    readAndPrintName();
    readAndPrintAge();
}
```

```
function readAndPrintName() : void
{
    var name : string = prompt("Enter your name: ", "");
    alert(`Your name is: ${ name }`);
}
```

```
function readAndPrintAge() : void
{
    var age : number = parseFloat(prompt("Enter your age: ", ""));
    alert(`Your age is: ${ age }`);
}
```

בדוגמא הנ"ל הפונקציה **readAndPrintNameAndAge** מפעילה את הפונקציה **readAndPrintName** ולאחר מכן את הפונקציה **readAndPrintAge** ולכן כאשר נפעיל אותה, שתי הפונקציות יופעלו אחת אחרי השנייה (לפי סדר הכתיבה).

קבלת פרמטרים לפונקציה

בכדי לאפשר לפונקציה יותר גמישות ניתן לכתוב פונקציה שתקבל פרמטרים. הפרמטרים הינם למעשה משתנים או ערכים אשר הפונקציה מקבלת. הפונקציה תקבל פרמטרים אלו ובהתאם לפרמטרים תבצע את פעולתה. כאשר נפעיל פונקציה המקבלת פרמטרים, נצטרך לשלוח אותם לפונקציה.
דוגמא:

```
var num : number = 8;
printNumbers(num);
printNumbers(5);

function printNumbers(x : number) : void
{
    var result="";
    for (var i = 1; i <= x; i++)
        result += i + ", ";

    alert(result);
}
```

פלט:

1,2,3,4,5,6,7,8,
1,2,3,4,5,

הפונקציה **printNumbers** מקבלת כפרמטר משתנה בשם x מסוג `number`. תפקידה של הפונקציה הוא להדפיס את כל המספרים מ-1 ועד x . בכל פעם שנקרא לפונקציה נקבל פלט אשר יתאים למספר ששלחנו. בפעם הראשונה שהופעלה הפונקציה, שלחנו לה משתנה אשר מכיל את הערך 8 (`num`) ובפעם השנייה את הערך 5.

הערך שנשלח לפונקציה נכנס לתוך המשתנה x המוגדר כפרמטר של הפונקציה ובכל פעם הפונקציה ביצעה את פעולתה, בהתאם לערך שהפרמטר x קיבל.
פונקציה יכולה לקבל גם יותר מפרמטר אחד. לדוגמא:

```
var str : string = "Stam string...";
var num : number = 3;
printString(str, num);

function printString(s : string, x : number) : void
{
    for (var i = 1; i <= x; i++)
        alert(i + ": " + s);
}
```

פלט (כל שורה בחלון שונה):

- 1: Stam string...
- 2: Stam string...
- 3: Stam string...

בדוגמא זו הפונקציה **printString** מקבלת משתנה **s** מטיפוס **string** המציג מחרוזת להדפסה, ומשתנה **x** מטיפוס **Number** המציג כמה פעמים להדפיס את **s**. מכיוון שכאשר הפעלנו את הפונקציה שלחנו לה את המחרוזת "Stam string..." ואת המספר 3, היא הדפיסה את המחרוזת 3 פעמים. ניתן לראות שבכדי לקבל פרמטרים לפונקציה, יש להצהיר עליהם בתוך הסוגריים שלאחר שם הפונקציה אך ללא המילה השמורה **let** או **var** (כמו בהצהרה על משתנה רגיל) עם הגדרת סוג המשתנה, כאשר הם מופרדים בפסיקים (,), ובכדי להשתמש (בתוך הפונקציה) בפרמטרים אלו יש לפנות אליהם באמצעות שמם.

כאשר נפעיל פונקציה המקבלת פרמטרים, חובה:

- לשלוח אליה את כל הפרמטרים (לא יותר או פחות).
- לשלוח את הפרמטרים לפי הסדר בו היא מקבלת אותם.
- כאשר נשלח לפונקציה ערך הוא חייב להיות מאותו הטיפוס של הפרמטר שהפונקציה מצפה לקבל.

אלו הם כללי חובה. במידה ולא נעמוד בהם לא נוכל להריץ את התוכנית, או שנקבל תוצאה לא צפויה.

החזרת ערך מפונקציה

פונקציות שכתבנו עד כה לא החזירו תשובות ולכן עשינו שימוש במילה השמורה **void**. שימוש בנקודתיים אחרי שם הפונקציה והפרמטרים מגדיר מה הסוג שאותו הפונקציה תחזיר.

בכדי לאפשר גמישות עוד יותר גדולה בשימוש בפונקציה, ניתן להחזיר ערך מפונקציה. אפשר להסביר ערך החוזר מפונקציה כתשובה לשאלה עליה הפונקציה צריכה לענות. פונקציה יכולה להחזיר ערך אחד בלבד או לא להחזיר ערך כלל (**void**).

בכדי להחזיר את הערך מהפונקציה יש להגדיר את סוג הערך החוזר לאחר שם הפונקציה והפרמטרים ויש לכתוב את המילה **return** ולאחריה את הערך שהיא צריכה להחזיר.

בכדי לקבל את הערך החוזר, יש לכתוב לפני הקריאה לפונקציה את שם המשתנה אליו יכנס הערך החוזר בתוספת אופרטור ההשמה (=).

לדוגמא:

```
var res : number, num : number = 5;
res = Sum(num);
alert(`Sum of 1 to ${num} is ${res}`);
```

```
function Sum(x : number) : number
{
    var s : number = 0;
    for (var i = 1; i <= x; i++)
        s += i;
    return s;
}
```

פלט:

Sum of 1 to 5 is 15

בדוגמא זו הפונקציה **Sum** צריכה להחזיר את סכום המספרים מ - 1 ועד ולמספר שנשלח אליה (פרמטר x). בסוף הפונקציה Sum ניתן לראות שהיא מחזירה את תוצאת החישוב אשר נמצא במשתנה **s**. **בקוד** ניתן לראות את הקריאה לפונקציה אשר שלחנו לה את **num** כפרמטר (אשר מכיל את הערך 5) ולפני הקריאה לפונקציה מצוין שהתשובה שתחזור מהפונקציה, תיכנס לתוך המשתנה **res**. כאשר נציג את תוכן המשתנה **res** למסך נראה שאכן הוא מכיל את תוצאת הפונקציה (15).

לא חובה להשתמש בערך החוזר מפונקציה. ניתן להפעיל פונקציה המחזירה ערך ולא לרשום לפני הקריאה את המשתנה שיקבל את הערך שיחזור (כמובן שלשלוח פרמטרים כן חייבים). זה אומר שיכולנו להפעיל את הפונקציה **Sum** גם כך:

```
Sum(num);
```

במקרה הספציפי הזה אין לכך משמעות, שכן ללא הערך החוזר אין צורך להפעיל את הפונקציה, אך ישנם מקרים בהם לא צריך את הערך החוזר.

פעולת החזרת ערך מפונקציה, למעשה מסיימת בו במקום את ריצת הפונקציה. כלומר, אם נכתוב את הפקודה **return** באמצע הפונקציה הערך יחזור וכל מה שכתוב לאחר ה- **return** לא יתבצע.

לדוגמא:

```
var big : number = getBig(2, 3, 1);  
alert(big + " is big");
```

```
function getBig(a : number, b : number, c : number) : number  
{  
    if (a > b && a > c)  
        return a;  
  
    if (b > c)  
        return b;  
  
    return c;  
}
```

פלט:

3 is big

הפונקציה **getBig** מקבלת 3 מספרים ומחזירה את הגדול. הבדיקה הראשונה בודקת האם **a** הוא הגדול (גדול מ- **b** וגם גדול מ- **c**) אם כן מחזירה אותו והפונקציה מסתיימת. אם הגענו לבדיקה השנייה (**b > c**) זה אומר ש- **a** אינו הגדול (אפילו שאין **else** ל- **if** הראשון) ולכן נותר לבדוק את **b** ו- **c**. אם **b** גדול מ- **c** יוחזר **b** והפונקציה תסתיים. אם הגענו לשורה האחרונה בפונקציה זה אומר ששתי הבדיקות לא היו נכונות (**false**) ולכן **c** הוא הגדול.

באמצעות **return** ניתן גם לסיים פונקציה אשר אינה מחזירה ערך.

לדוגמא:

```
var str : string = "";
var num : number = 3;
printString(str, num);
```

```
function printString(s : string, x : number) : void
{
    if (s == "" || x <= 0)
        return;

    for (let i = 1; i <= x; i++)
        alert(i + ": " + s);
}
```

הפונקציה **printString** מדפיסה מחרוזת מסוימת (s) מספר פעמים (x). אם המחרוזת ריקה או שמספר הפעמים קטן או שווה 0, אין צורך להיכנס ללולאה וניתן לסיים את הפונקציה, או במילים אחרות לצאת מהפונקציה.

שמות פונקציות

הכללים למתן שמות לפונקציות הינם כמו הכללים למתן שמות למשתנים (ראה "כללי מתן שמות למשתנים" בפרק "משתנים").

זאת למעט כלל אחד: שמות של פונקציות יתחילו באות קטנה ולא באות גדולה. כאשר שם של פונקציה מכילה מספר מילים כל מילה תתחיל באות גדולה (כמו במשתנים).

טווח הכרה של משתנים

סוגי משתנים:

1. משתנים לוקליים (מקומיים):
משתנים המוגדרים בתוך ה- { } של פונקציה כלשהי. המשתנים מוכרים רק בתוך הפונקציה ו"מתים" ברגע שהפונקציה מסתיימת, כלומר שטח הזיכרון שהם תפסו משתחרר. בקריאה הבאה לפונקציה הם יוגדרו מחדש ויוקצה להם מקום חדש בזיכרון.
2. משתנים פורמאליים (פרמטרים):
הם משתנים המוגדרים בחתימת הפונקציה. משתנים אלו מתנהגים בדיוק כמו המשתנים **הלוקליים** למעט העובדה שהם מאוחילים ע"י הערכים שנשלחים לפונקציה.
3. משתנים גלובליים:
הם משתנים המוגדרים מחוץ לכל הפונקציות. משתנים אלו מוכרים ע"י כל הפונקציות ו"מתים" רק בסיום התוכנית. נשתמש במשתנים גלובליים רק כאשר נהיה חייבים ולא סתם כך מפני שזה נוח, מכיוון שהשימוש בהם "פותח את הדלת" לתקלות.
בכל בלוק של { } ניתן להגדיר משתנים, גם בבלוקים של מבני בקרה (for, if וכו'). משתנה מוכר מאותו מקום שבו הוגדר ועד לסיום הפונקציה. כאשר הפונקציה מסתיימת המשתנה "אינו פעיל" יותר (וסביר להניח שתא הזיכרון שלו ישמש בהמשך למשתנה אחר ש"יוולד" במקומו).
משתנים פורמאליים של פונקציה (הפרמטרים) שהיא מקבלת נחשבים כאילו הוגדרו בתוך הבלוק של הפונקציה, כלומר משתנים לוקליים של הפונקציה.

למשל, ניסיון למשתנים **pow**, **x** אשר מוגדרים בפונקציה **power** יגרור שגיאה:

```
alert(x); //Error
alert(pow); //Error
function power(x : number) : void
{
    let pow = x * x;
    alert(pow);
}
```

ניתן גם להגדיר 2 משתנים (או יותר) בעלי אותו השם בפונקציות שונות, מכיוון שכל אחד מהם הוא משתנה אחר.

התוכנית שלהלן חוקית לחלוטין, למרות שב- 2 פונקציות (print1, print2) הוגדרו אותם שמות של משתנים:

```
print1(17);
print2(17);
```

```
function print1(x : number) :void
{
    let y : number = 1;
    let sum : number = x + y;
    alert(`${x} ${y} ${sum}`);
}
function print2(x : number) : void
{
    let y : number = 1;
    let sum : number = x + y;
    alert(`${x} ${y} ${sum}`);
}
```

כאשר מעבירים משתנה לפונקציה מתבצעת העתקת תוכן המשתנה, לתוך המשתנה הפורמאלי של הפונקציה. כל שינוי שנבצע במשתנה של הפונקציה, לא ישפיע על המשתנה הנשלח, מפני שהמשתנה של הפונקציה, הינו עותק אחר של המשתנה הנשלח.
לדוגמא:

```
var a : number =10, b : number =20;
alert('Before swap: a = ${a}, b = ${b}');
swap(a, b);
alert('After swap: a = ${a}, b = ${b}');

function swap(x: number, y: number) : void
{
    var tmp: number = x;
    x = y;
    y = tmp;
}
```

פלט:

Before swap: a = 10, b = 20
After swap: a = 10, b = 20

בדוגמא הנ"ל הפונקציה **swap** אמורה להחליף בין המשתנים שנשלחים אליה. לפי הפלט אנו רואים שפונקציה זו לא מבצעת את עבודתה, מכיוון שהחלפת המשתנים שנעשתה בפונקציה, נעשתה על עותק משוכפל שלהם וכאשר הפונקציה סיימה את פעולתה העותקים "מתו" והמשתנים המקוריים שנשלחו לא השתנו.

שימוש בפונקציות קימות

Typescript מספקת המון פונקציות קיימות, אשר נוכל להשתמש בהן. השימוש בפונקציות אלו חוסך מאיתנו זמן, שכן במקום לכתוב כל דבר בעצמנו, אנו יכולים להשתמש בפונקציות שכבר כתבו בשבילנו. למעשה בכל תוכנית שאנו כותבים, אנו משתמשים בפונקציות אלו. כאשר נרצה למצוא פונקציות קיימות, נוכל לחפש אותן במערכות עזרה ותיעוד ובאינטרנט.

לדוגמא:

הפונקציה `parseInt` המקבלת מחרוזת ומחזירה את מה שכתוב במחרוזת כטיפוס מספרי (Number).
הפונקציה `alert()` המקבלת מחרוזת ומציגה אותה של המסך.
ועוד...

סיכום

ניתן לראות את הפונקציה כקופסא שחורה, עצמאית, המבצעת פעולה כלשהי. הפעולה תהיה פעולת חישוב, ציור, הדפסה, רישום מידע, בדיקת מידע, קליטת מידע ועוד אלפי פעולות. רצוי כי כל פונקציה תהיה אחראית על פעולה אחת, כך נוכל לשלוט טוב יותר בתוכנית ולטפל בכל נושא באופן נקודתי. יתרון נוסף בנית פונקציה המטפלת בנושא אחד בלבד, הנו שימוש חוזר בפונקציה (ReUse), כלומר יתכן מאוד שהנושא המטופל יחזור בתרגיל הבא, בתוכנית הבאה, בפרויקט הבא, במוצר הבא. כך נוכל פשוט להעביר את הפונקציה לתרגיל הבא, לאחר שבדקנו ועבדנו איתה בתרגיל הנוכחי ולהפעילה בעת הצורך.

ההגדרה קופסא שחורה פשוטה כמשמעה, המשתנים המוגדרים בפונקציה מוכרים רק לה ואינם נגישים לשום גורם בתוכנית, כך נוכל להבטיח "קוד בטוח" כי המשתנים משתנים אך ורק בפונקציה עצמה. הקשר של הפונקציה לתוכנית הנו הפרמטרים שהיא מקבלת, מומלץ שהפונקציה תקבל מעט פרמטרים כך הקשר שלה לתוכנית יהיה "בעל צימוד חלש" והסיכוי לתקלות יקטן. מעתה ואילך כל תוכנית שנכתוב תכלול פונקציות, תתבסס על הפונקציות הקיימות בסביבת העבודה ועל פונקציות שנכתוב אנו.

תרגילים:

1. בנה פונקציה המקבלת 2 מספרים ומחזירה את הגדול מביניהם.
2. בנה פונקציה המקבלת 2 מספרים ומחזירה את התוצאה של הראשון בחזקת השני.
3. בנה פונקציה המקבלת מספר ומחזירה את העצרת שלו.
4. בנה פונקציה המקבלת מספר ומחזירה אותו בסדר הפוך.
5. בנה פונקציה המקבלת הודעה ומספר: הפונקציה תדפיס את ההודעה למשתמש, תקלוט כמות מספרים לפי המספר שקיבלה ותחזיר את סכום המספרים.
6. בנה פונקציה המקבלת 3 מספרים ומחזירה את הגדול מביניהם. בתוך הפונקציה יש להשתמש בפונקציה מהתרגיל הראשון על מנת למצוא את המספר הגדול מבין שלושת מספרים. (אין לשנות את הפונקציה מהתרגיל הראשון, נשתמש באותה פונקציה בדיוק.)
7. כתוב פונקציה המקבלת שני פרמטרים המציינים תחום מספרי כלשהו, הפונקציה תחזיר מספרי אקראי בתחום המספרים
8. כתוב פונקציה המקבלת מספר ומציגה שורת כוכביות באורך המספר.
9. כתוב פונקציה המקבלת מספר n ומציגה כוכביות באופן הבא:
בהנחה ש- $n = 4$

```
*
**
*
***
**
*
****
***
**
*
```

10. כתוב פונקציה המקבלת מספר n והצג את התבנית הבאה:
בהנחה ש- $n = 4$

```
4444****4444****4444****4444****
333***333***333***
22**22**
1*
```

פרק 8 - מערכים

בפרק זה נלמד את הנושאים הבאים:

- מהו מערך
- שימוש במערך
- שיטות לאתחול מערך
- העברת מערך לפונקציה
- מיון מערך
- חיפוש במערך
- מהו מערך דו - ממדי (מטריצה)
- שימוש במערך דו-ממדי

מהו מערך

נניח ונרצה לקלוט מהמשתמש 100 מספרים ולאחר הקליטה להדפיס אותם בסדר הפוך. באמצעות הכלים שהכרנו עד כה נצטרך להגדיר 100 משתנים בזיכרון שיכילו את המספרים. ואם נרצה לקלוט 10000 מספרים, האם נגדיר 10000 תאים בזיכרון? פעולה כזו תהפוך את הקוד למסורבל ובלתי ניתן לתחזוקה. בכדי לפתור בעיות מסוג זה נשתמש במערך.

מערך – רצף של משתנים בזיכרון.

מערך הינו מספר תאים רציפים בזיכרון, כאשר נגדיר את המערך נוכל להגדיר בנוסף כמה תאים הוא יכיל. כל תא במערך (שהוא בעצם משתנה לכל דבר) מקבל מספר המציין את מיקומו, מספר זה מכונה אינדקס התא. כאשר נרצה לפנות לתא מסוים במערך, נפנה אליו דרך שם המערך ובנוסף נציין את אינדקס האיבר אליו נרצה לגשת.

חשוב לדעת - Javascript ובTypescript המערך הוא דינמי ומאפשר הסרה והוספה של ערכים. מערכים בשפות אחרות נעולות לגודל שבה הם נוצרו ולכן לא ניתן להוסיף ערך מעבר לכמות שהוגדרה בהתחלה

שימוש במערך

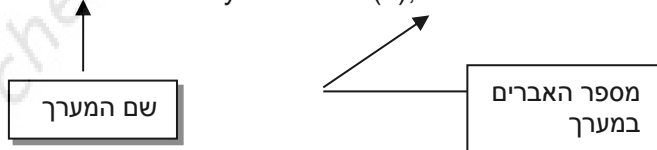
קיימות מספר אופציות ליצירת מערך :

אפשרות 1

יצירת מערך:

<<

`var arr = new Array<number>(5);`



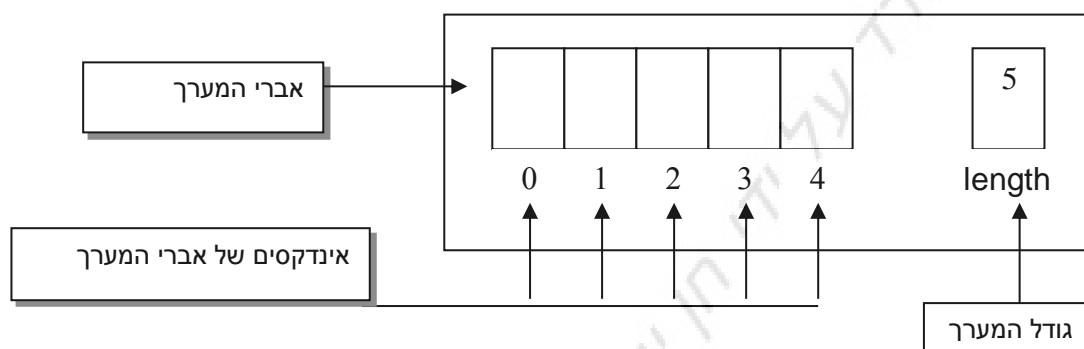
בדוגמאות אלו אנו עוסקים בעיקר מערך של מספרים מסוג number

אפשרות 2

```
var arr: number[] = [];
```

בדוגמא זו לא הגדרנו גודל מסוים למערך.

בדוגמא זו הוגדר מערך המכיל 5 תאים.
ציור זיכרון של המערך:



ניתן להבין מהציור שהאיבר הראשון במערך מקבל את האינדקס 0, השני את האינדקס 1 וכן הלאה, כאשר האינדקס של האיבר האחרון במערך הוא גודל המערך (length) פחות 1.
ניתן להגדיר בסוגריים המציינים את גודל המערך גם שם של משתנה מספרי.

לדוגמא:

```
var size = 10;  
var arr = new Array<number>(size);
```

בקטע הקוד הנ"ל הוגדר משתנה בשם size המכיל 10, ובאמצעות משתנה זה הוגדר גודל המערך.

גישה לאיבר מסוים במערך:

בכדי לגשת לאיבר במערך יש לפנות אליו דרך שם המערך ולאחר השם יש לציין את האינדקס של האיבר בתוך סוגריים מרובעים [].

```
arr[0] = 8; //גישה לאיבר הראשון במערך  
arr[3] = 13; //גישה לאיבר הרביעי במערך
```

```
alert(arr[3].toString()); //יודפס 13 – האיבר הרביעי במערך
```

ניתן גם לדעת את גודל המערך באמצעות המשתנה length שהמערך מכיל:

```
alert(arr.length.toString()); //יודפס 10 – גודל המערך
```

חשוב להבין שכאשר ניגש לאיבר מסוים במערך, כל הביטוי המציין את האיבר עצמו מתנהג כמו משתנה בודד.

arr[3] – הינו משתנה אחד בודד מטיפוס Number, ולכן ניתן לבצע על איבר זה כל פעולה שניתן לבצע על משתנה מטיפוס number, כולל השמה לתוכו, ביצוע פעולות חשבון, קבלת הערך למשתנה אחר, העברה לפונקציה, וכו'.

כאשר נרצה לבצע פעולה מסוימת על כל המערך, נשתמש בד"כ בלולאת מונה, כך שמונה הלולאה יציין בכל איטרציה של הלולאה את אינדקס האיבר במערך.

לדוגמא:

```
var arr = new Array<number>(7);
```

```
for (let i = 0; i < arr.length ; i++)
{
    arr[i] = i * 2;
}
```

```
var result : string = "";
for (let i = 0; i < arr.length ; i++)
{
    result += arr[i] + ", ";
}
```

```
alert(result);
```

פלט:

0,2,4,6,8,10,12,

בדוגמא הנ"ל הוגדר מערך בעל 7 תאים. באמצעות לולאת for ניתן לרוץ החל מהאיבר ה- 0 ועד 1 פחות מגודל המערך (סה"כ 7 איטרציות). כאשר בלולאה הראשונה לכל איבר במערך, יוכנס $i \cdot 2$ (מספרים זוגיים החל מ- 0). ניתן לראות שבתוך הסוגריים המרובעים המציינים את אינדקס האיבר, מופיע i כאשר בכל איטרציה של הלולאה i גדל באחד והדבר יביא לכך שבכל איטרציה נפנה לאיבר הבא במערך. הלולאה השנייה עוברת על כל המערך ומוסיפה את כל האברים שבמערך למשנה מסוג String (בתוספת פסיק). ניתן לראות שאכן הפלט תואם למידע שהוכנס לתוך המערך.

כלל חשוב: כאשר נרוץ בלולאה על מערך, לעולם לא נכתוב במפורש את גודל המערך (כתנאי עצירה לסיום הלולאה), אלא נשתמש במשתנה length של המערך בכדי לדעת מה גודלו. זאת בכדי שהלולאה תתאים לכל גודל של מערך, גם אם נחליט לשנות את גודלו ביום מן הימים.

לאחר יצירת מערך ניתן לשנות את גודלו:

```
var arr = new Array(7);
for (var i = 0; i < arr.length ; i++)
{
    arr[i] = i * 2;
}
arr[10] = 5;
```

בדוגמא זו גודלו של המערך יהפוך להיות 11 (האיבר באינדקס 10 הוא האיבר ה-11) וכן האיברים באינדקסים 7 – 9 יהיו ריקים.

 המערך יכול להכיל איברים מטיפוסים שונים, אך בד"כ לא מומלץ לבצע זאת, מפני שכאשר נרצה לבצע פעולה מסוימת על כל המערך, עלולות להיות תקלות.

שיטות לאתחול מערך

ישנן מספר אפשרויות לאתחול מערך.

אפשרות 1: אתחול מערך ללא ציון גודל וללא ציון ערך לאיברים:

```
var arr = new Array<number>();
```

כאשר נגדיר מערך בצורה זו, הדפדפן ייצור מערך אשר אינו מכיל אברים (אורך 0). כמובן שניתן לאחר מכן לשנות את גודלו.

אפשרות 2: אתחול מערך עם ציון גודל במפורש, ללא ציון ערך לאיברים:

```
var arr = new Array<number>(5);
```

כאשר נגדיר מערך בצורה זו, כל האיברים במערך יאותחלו לערך null.

אפשרות 3: אתחול מערך ללא ציון גודל במפורש וציון ערכים התחלתיים לאיברים.

```
var arr = new Array<number>(5, 8, -4, 6, -2);
```

בצורה זו, האיברים במערך יקבלו את הערכים המופיעים בסוגריים. כאשר האיבר הראשון יקבל את הערך הראשון, השני את הערך השני וכו'. גודל המערך ייקבע אוטומטית לפי מספר הערכים המופיעים בסוגריים.

העברת מערך לפונקציה

כאשר נרצה לקבל מערך לפונקציה כפרמטר, נצהיר עליו בסוגריים של חתימת הפונקציה. ההצהרה דומה מאד לקבלת משתנה רגיל לפונקציה. בתוך הפונקציה נפנה אליו כמו מערך רגיל (שהוגדר בתוך הפונקציה).

לדוגמא:

```
function FuncName(myArr : number[]) // same as: function FuncName(myArr :  
Array<number>)  
{  
}  
}
```

כאשר נרצה לשלוח את המערך לפונקציה, נכתוב בסוגריים של הקריאה לפונקציה את שמו של המערך.

לדוגמא:

```
var arr : Array<number> = new Array<number>(5);
FuncName(arr);
```

כידוע, כאשר מעבירים משתנה רגיל לפונקציה, נוצר עותק של המשתנה בתוך הפונקציה, וכאשר נשנה את העותק בתוך הפונקציה, המשתנה המקורי (זה ששלחנו לפונקציה) לא ישתנה. לא כך במערכים. כאשר נעביר מערך לפונקציה, הוא לא עובר העתקה, והפונקציה עובדת על אותו העותק בזיכרון. לכן, כאשר נשנה את המערך בתוך הפונקציה, השינוי יחול גם על המערך המקורי (זה ששלחנו לפונקציה).

דוגמא להעברת מערך לפונקציה:

```
Start();
```

```
function Start()
{
    var arr : Array<number> = new Array<number>(5);
    InitArray(arr);
    ShowArray(arr);
}
```

```
function InitArray(arr: Array<number>)
{
    for (let i = 0; i < arr.length ; i++)
    {
        arr[i] = i;
    }
}
```

```
function ShowArray(arr : Array<number>)
{
    alert(arr.toString());
}
```

פלט:

0,1,2,3,4

ניתן לראות שהמשתנה arr הוגדר בפונקציה Start והועבר לפונקציה InitArray שתפקידה לאתחל את המערך כך שכל איבר במערך יכיל מספר השווה לאינדקס שלו. לאחר מכן המערך נשלח לפונקציה ShowArray, שתציג אותו. ניתן לראות שהפונקציה ShowArray הציגה את אותו המערך שאותחל בפונקציה InitArray, מה שאומר ששתי הפונקציות עובדות על אותו המערך ולא כל אחת עובדת על עותק אחר, כמו שקורה במשתנים רגילים.

מיון מערך

ישנם מספר אלגוריתמים המיועדים לביצוע מיון של מערך.

- מיון ליניארי
- מיון בועות
- ועוד

שלבים במיון ליניארי:

1. השוואת האיבר הראשון מול כל האיברים שאחריו.
2. במידה ונמצא שאיבר מסוים קטן יותר מהאיבר הראשון – ביצוע של החלפה ביניהם.
3. לאחר סיום ההשוואה מול כל האיברים בטוח שאיבר הקטן ביותר נמצא במקומו.
4. חזרה לשלב 1 – (השוואת האיבר השני מול כל האיברים שאחריו) וכו' עד שמגיעים לאיבר האחרון.

שלבים במיון בועות:

1. מעבר על כל המערך והשוואה בין שני איברים צמודים (הראשון והשני, השני והשלישי וכו') ו"דחיפת" הגדול כלפי מעלה באמצעות החלפה.
2. לאחר סיום מעבר אחד על המערך בטוח שהאיבר הגדול ביותר נמצא בסוף.
3. חזרה לשלב 1 – ועצירת ההשוואה איבר אחד לפני הסוף וכו'.

חיפוש במערך

כאשר נרצה לחפש איבר מסוים במערך, יש לרוץ על כל המערך עד שמוצאים את האיבר, או מגיעים לסוף ובכך יודעים שאיבר לא נמצא. במידה והמערך ממין, ניתן "לחסוך" בחיפושים ולהשתמש בשיטת חיפוש הנקראת "חיפוש בינארי".

שיטת חיפוש בינארי, מזכירה חיפוש של שם בספר טלפונים. במקום לעבור שם אחר שם, עד שנמצא את השם המבוקש, נפתח באמצע הספר ונחליט האם השם המבוקש נמצא לפני או אחרי השם הנוכחי הנמצא בספר, ואז נבחר בחצי הספר שבו השם המבוקש יכול להיות ושוב נחלק אותו באמצע ונחליט האם השם המבוקש נמצא לפני/אחרי השם הנוכחי בספר וכן הלאה עד שנמצא את השם המבוקש.

בשיטת החיפוש הבינארי מחזיקים שני משתנים:

start – המייצג את התחלת האזור בו האיבר יכול להימצא.

end – המייצג את סוף האזור בו האיבר יכול להימצא.

שלבים בשיטת החיפוש הבינארי:

1. אתחול המשתנים start, end לאיבר הראשון ואחרון במערך (בהתאמה).
2. מציאת האמצע בין start ל-end ובדיקה האם האיבר המבוקש נמצא.
 - a. אם כן – סיום החיפוש.
 - b. אם לא – בדיקה האם האיבר המבוקש נמצא מימין או משמאל לאמצע.
 - i. אם מימין העברת ה-start לאיבר אחד אחרי האמצע.
 - ii. אם משמאל העברת ה-end לאיבר אחד לפני האמצע.
 - c. חזרה לשלב 2 – עד אשר ה-start יהיה יותר גדול מה-end.

מהו מערך דו-ממדי (מטריצה)

עבור תוכנה כלכלית נדרש להחזיק את הטבלה הבאה:

	Jan	Feb	Mar	Apr	Ma	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Tot
96													
97													
98													
99													
00													
01													
02													
03													
04													
05													
06													

לכל שנה בעשור האחרון יוחזקו 12 המדדים החודשיים ובנוסף יוחזק המדד השנתי. למעשה מה שאנחנו צריכים הוא מערך של 10 מערכים בגודל 13 כל אחד. בכל מערך של 13 תוחזק שנה אחת כאשר בתא האחרון יושב המדד השנתי ובתאים 0-11 יושבים מדדי החודשים.

בכדי לפתור בעיה כזו, עם היכולות המוכרות לנו עד עתה, עלינו ליצור 10 מערכים. ישנה דרך יותר פשוטה לפתור בעיה זאת והיא להשתמש במערך דו-ממדי.

מערך דו-ממדי דומה מאד בתפיסתו למערך חד-ממדי. ההבדל ביניהם הוא שכאשר ניצור מערך חד-ממדי תיווצר שורה אחת של תאים בזיכרון, וכאשר ניצור מערך דו-ממדי ייווצרו בזיכרון מספר שורות. מערך דו-ממדי מזכיר מאד טבלה עם שורות ועמודות ומכונה גם מטריצה.

שימוש במערך דו-ממדי

בכדי ליצור מערך דו-ממדי ב-JavaScript יש ליצור מערך חד-ממדי (מערך ראשי), ובכל איבר במערך החד-ממדי יש להכניס מערך חד-ממדי (מערכי משנה).

יצירת מערך דו-ממדי בגודל 3 שורות ו-4 עמודות:

```
var mat: Array<Array<number>> = new Array<Array<number>>(3);
mat[0] = new Array<number>(4);
mat[1] = new Array<number>(4);
mat[2] = new Array<number>(4);
```

בשורה 1 – נוצר מערך בשם mat (מערך ראשי) בגודל 3 (אינדקסים 0 – 2)
בשורה 2 – בתוך כל איבר במערך הראשי נוצר מערך משני בגודל 4
סה"כ: $3 * 4 = 12$ תאים

ניתן גם לבצע זאת באמצעות לולאה:

```
var mat: Array<Array<number>> = new Array<Array<number>>(3);
for(let i=0;i<mat.length;i++)
    mat[i] = new Array<number>(4);
```

גם בדוגמא זו נוצרה מטריצה בגודל $4 * 3$, אך בשונה מהדוגמא הקודמת, עברנו בלולאה על המערך הראשי ובתוך כל איבר הוכנס מערך בגודל 4. כמובן ששיטה זו עדיפה, ומקצרת את הקוד מטריצות עם הרבה שורות.

- בדומה למערך חד-ממדי, גם במטריצה ניתן להגדיר את הגודל באמצעות משתנה.
- ניתן גם להגדיר גודל שונה לכל שורה במטריצה.

בכדי לגשת לאיבר מסוים במטריצה, יש לציין את מספר השורה וגם את מספר התא (בתוך השורה) אליו רוצים לגשת. במטריצה לכל שורה יש אינדקס ולכל עמודה יש אינדקס, כאשר אינדקסים אלו מתחילים מ-0 (כמו במערך חד-ממדי).

```
mat[1][2] = 20; // גישה לשורה 2 תא 3
alert(mat[1][2].toString()); // יוצג 20 – הצגת האיבר
```

בשורות קוד אלו, ישנה השמה של הערך 20 לתוך התא הנמצא בשורה השנייה (אינדקס 1) ובעמודה השלישית (אינדקס 2) ולאחר מכן הצגת הערך על תיבת alert.

בדומה למערך חד-ממדי, גם במטריצה חשוב להבין שכאשר ניגש לאיבר מסוים במטריצה, כל הביטוי המציין את האיבר עצמו מתנהג כמו משתנה בודד. לדוגמא אם הוכנס לתוך המשתנה `mat[0][0]` ערך מטיפוס `Number`, `mat[0][0]` יהיה משתנה אחד בודד מטיפוס `Number`, ולכן ניתן לבצע עליו כל פעולה שניתן לבצע על משתנה בודד מטיפוס `Number`, כולל השמה לתוכו, ביצוע פעולות חשבון, קבלת הערך למשתנה אחר, העברה לפונקציה, וכו'.

בכדי לבצע פעולה מסוימת על מטריצה, בד"כ נשתמש בלולאות מקוננות. הלולאה החיצונית תרוץ על השורות במטריצה והפנימית תעבור על כל התאים בשורה מסוימת. בכדי לדעת מה הגודל של כל מימד, נשתמש ב `length`. את המונים של הלולאות, נכתוב בסוגריים המרובעים [] המציינים את מיקום התא אליו ניגש.

לדוגמא:

```
var mat = new Array<Array<number>>(10);

for(let i=0;i<mat.length;i++)
    mat[i] = new Array<number>(5);

for (let i = 0; i < mat.length; i++)
{
    for (let j = 0; j < mat[i].length; j++)
    {
        mat[i][j] = i*j;
    }
}
```

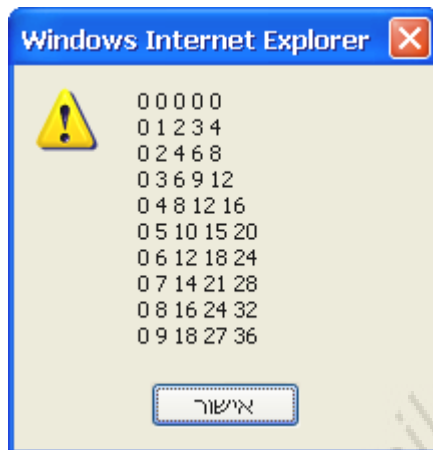
```
var result : string = "";

for (let i = 0; i < mat.length; i++)
{
    for (let j = 0; j < mat[i].length; j++)
    {
        result += mat[i][j] + " ";
    }

    result += "\n";
}

alert(result);
```

פלט:



בדוגמא הנ"ל ישנן שתי קבוצות של לולאות:

קבוצה ראשונה עוברת על כל המטריצה ומאתחלת אותה במספרים אקראיים. כאשר הלולאה החיצונית מונה את השורות ותנאי העצירה שלה הוא כל עוד i (מונה הלולאה) קטן ממספר השורות. הלולאה הפנימית עוברת על כל התאים בשורה ותנאי העצירה שלה הוא כל עוד j (מונה הלולאה) קטן ממספר העמודות שזה למעשה אורך המערך המשני. כאשר נפנה לתאים בתוך הלולאה, נעשה זאת באמצעות המונים של שתי הלולאות (i, j) .

קבוצה שנייה עוברת על כל המטריצה באותה השיטה בדיוק כמו בקבוצת הלולאות הראשונה, אך במקום להכניס מידע למטריצה, היא משרשרת את המידע למחרוזת. ניתן לשים לב שהשרשור בתוך הלולאה הפנימית הוא ללא ירידת שורה, ובלולאה החיצונית בסיום שרשור כל שורה, ישנו שרשור של ירידת שורה במסך.



תרגילים:

1. כתוב פונקציה המחזירה את האינדקס של המספר הגדול ביותר במערך.
2. כתוב פונקציה המחזירה את האינדקס של המספר הקטן ביותר במערך.
3. כתוב פונקציה הבודקת האם המערך סימטרי (פולינדרום).
4. כתוב פונקציה הבודקת האם המערך מסודר בסדר עולה.
5. כתוב פונקציה המחזירה את ממוצע איברי המערך.
6. כתוב פונקציה ההופכת את הסדר המערך, כך שיהיה מסודר מהסוף להתחלה.
7. אתחל מערך בגודל SIZE בערכים אקראיים שבין 0 ל-9, מצא: את הספרה השכיחה ביותר ואת הספרה הנדירה ביותר.
8. אתחל מערך בגודל SIZE בערכים אקראיים בין 1000 ל-9999, מצא: את הספרה השכיחה ביותר ואת הספרה הנדירה ביותר.
9. אתחל מערך בגודל SIZE בערכים אקראיים, כך שכל ערך יופיע פעם אחת בלבד.
10. נתונים שני מערכים המאותחלים בערכים אקראיים: האחד בן 50 איברים והשני בן 5 איברים. בדוק האם המערך הקטן מוכל בשלמותו בתוך הגדול וכמה פעמים.
11. נתונים שני מערכים באורך SIZE המאותחלים בערכים אקראיים בין 0 ל-9. חפש את רצף הערכים הזהה המכסימלי הקיים בשניהם.
לדוגמה:
8 5 3 0 3 1 8 2 8 7 3 5 4 0 8 5 7 3 1
1 8 4 0 2 9 4 3 6 9 2 3 1 8 2 8 7 3 3
12. קלוט מספר וחשב כמה ספרות שונות יש בו. לדוגמה: 4589655236 התשובה היא 7. ניתן להשתמש במערך עזר.
13. נתונים שני מערכים ממוינים באורך SIZE ומערך אחד באורך שני SIZE. העבר את ערכי שני המערכים למערך הגדול, באופן כזה שהמערך יהיה ממוין.
14. בנה פונקציה המקבלת מערך וממיינת אותו בשיטת מיון ליניארי.
15. בנה פונקציה המקבלת מערך וממיינת אותו בשיטת מיון בועות.
16. נמק מדוע עדיפה שיטת מיון הבועות על פני שיטת המיון הליניארי.
17. בנה פונקציה המקבלת מערך ומספר, ממיינת את המערך, ומחזירה את האינדקס של האיבר בו נמצא המספר. במידה והמספר לא נמצא יוחזר -1. חיפוש האיבר יבוצע באמצעות שיטת החיפוש הבינארי.

מטריצה:

1. אתחל מטריצה בגודל SIZE במספרים אקראיים ומצא:
 - את המספר הגדול ביותר.
 - את המספר הקטן ביותר.
 - את הממוצע.
 - את השורה עם הממוצע הגבוה ביותר.
 - את הטור עם הממוצע הקטן ביותר.
2. אתחל מטריצה בגודל SIZE במספרים אקראיים ובדוק האם מתקיים מצב חוקי. במצב חוקי מספר לא יופיע פעמיים באותה שורה או טור.