

פונקציות

הפונקציות הן אבני הבניין של קוד קריא, ניתן לתחזוקה וניתן לשימוש חוזר. פונקציה היא בלוק קוד שנועד לביצוע משימה מסוימת. לאחר שהוגדרה, ניתן לקרוא לה מהתוכנית הראשית או מפונקציות אחרות ולעשות בה שימוש חוזר.

במסמך הבא נעבור על חלקים חשובים שיש לחדד בנושא הפונקציות:

- טווח הכרה ואורך חיים של משתנים בפונקציה
- החזרת ערך מפונקציה
- דוגמאות לשגיאות אפשריות בפונקציה

טווח הכרה ואורך חיים של משתנים בפונקציה

טווח הכרה	
כל הפונקציה	var
רק בתוך הבלוק בו המשתנה הוגדר, ורק מהשורה של הגדרת המשתנה	let

Block scoping

let מאפשר ליצור משתנים שהם *block-scoped* - נגישים רק בתוך הבלוק הפנימי המקיף אותם. הקוד הבא מדגים כי let מגדיר את המשתנה שמוכר רק בתוך הבלוק של משפט if:

```
function func():void {  
  if (true) {  
    let tmp:number = 123;  
  }  
  alert(tmp); // Error: tmp is not defined - compilation error  
}
```

var לעומת זאת, מאפשר ליצור משתנים שהם *function-scoped* – נגישים בכל שטח הפונקציה

```
function func():void {  
  if (true) {  
    var tmp:number = 123;  
  }  
  alert (tmp); // 123  
}
```

Block scoping מאפשר ליצור משתנים בתוך הבלוק הפנימי המקיף אותם, בעלי שם זהה למשתנים שמוגדרים בבלוק החיצוני, ובכך ליצור מעין "צל" שיגרום לכך שבכל בלוק ההתייחסות תהיה למשתנה שהוגדר בו.

```
function func():void {  
  let foo:number = 5;  
  if (true) {
```

```

    let foo:number = 10; // shadows outer `foo`
    alert(foo); // 10
  }
  alert (foo); // 5
}

```

החזרת ערך מפונקציה

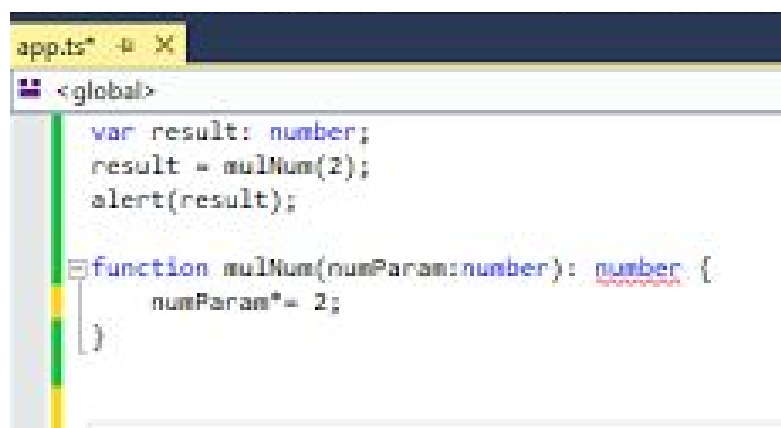
כאשר ניצור פונקציה, שהגדרנו שהיא מחזירה ערך מטיפוס מסויים (לא void), עלינו לדאוג שהבלוק שמכיל את תוכן הפונקציה יחזיר ערך בכל מצב

- כדי להבין את העניין, נחזור בקצרה על 3 סוגי השגיאות האפשריות בתוכנית, ועל ההשלכות שלהן:
- שגיאת קומפילציה - שגיאה שמתרחשת בעקבות תחביר שגוי שלא עומד בכללים של השפה. השלכה: התוכנית לא תוכל להתחיל לרוץ כאשר קיימת שגיאת קומפילציה
 - שגיאת זמן ריצה - שגיאה המתרחשת בעקבות פעולה שגורמת לשגיאה בזמן הריצה (לדוגמא: לולאה אינסופית, שגורמת בסופו של דבר לקריסת התוכנית)
 - שגיאה לוגית - התוכנית תרוץ ללא קריסה, אולם התוצאה שהלקוח יקבל לא תהיה התוצאה הנכונה, והלוגיקה המבוקשת לא תמומש לפי הצורך

כעת, נוכל לחזור לנושא הפונקציות, ולהבין בעזרת דוגמא פשוטה אילו שגיאות עלולות להתרחש כאשר המתכנת לא ידאג להגדיר את הערך המוחזר לכל מצב.

ניצור תוכנית פשוטה, המכילה פונקציה בשם mulSum, תפקיד הפונקציה לקבל מספר ולהחזיר את מכפלתו.

נתון קטע הקוד הבא:



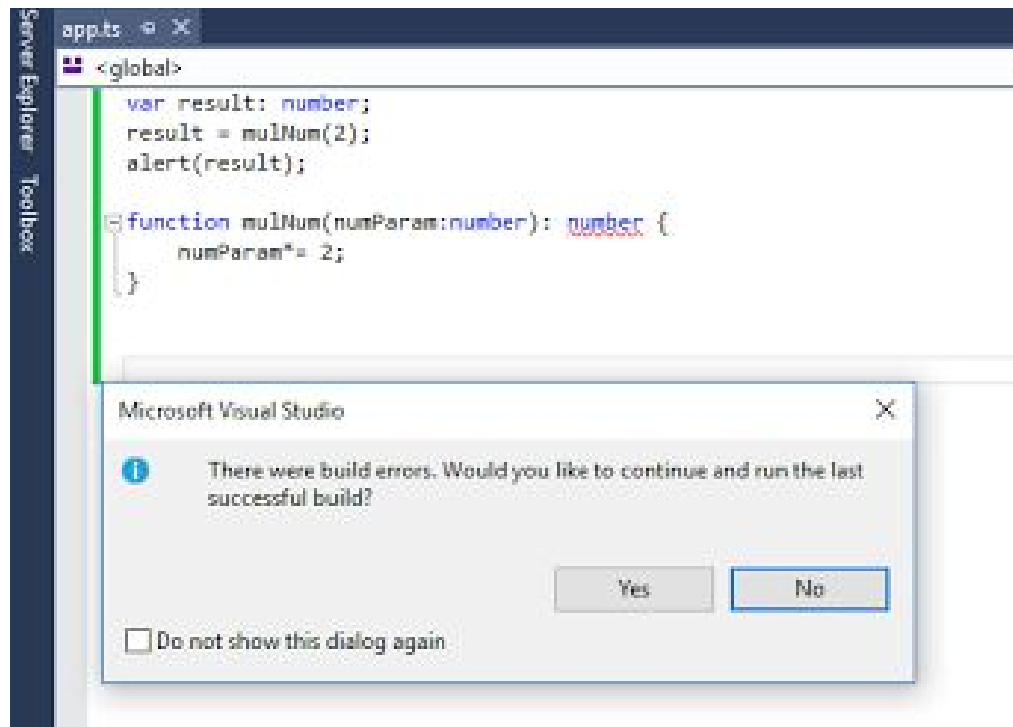
```

app.ts  X
<global>
var result: number;
result = mulNum(2);
alert(result);

function mulNum(numParam:number): number {
  numParam*= 2;
}

```

אנו יכולים מיד לראות שקיימת בעיה בתוכנית, ויש סימון שגיאה מודגש. ננסה בכל זאת להריץ את התוכנית, ונקבל את התוצאה הבאה:



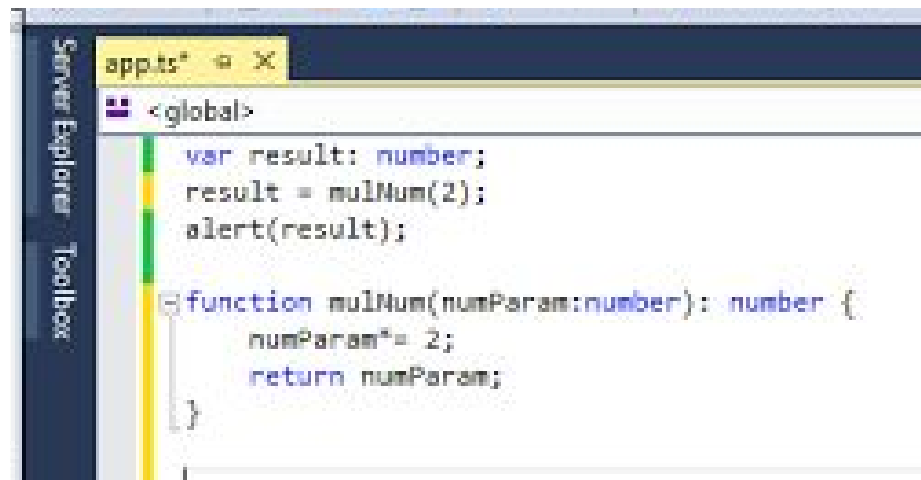
כלומר, קיבלנו הודעה שקיימת שגיאת קומפילציה (build error), ולכן התוכנית שלנו לא יכולה לרוץ לפי הגירסה הזו של הקוד המכיל תחביר שגוי.

כעת, נבדוק בדיוק מהי הודעת השגיאה, ע"י מעבר עם העכבר על השגיאה המסומנת באדום, השגיאה שאנו נראה היא: "Function declared a non-void return type, but has no return expression"



מסקנה: כאשר יצרנו פונקציה שמחזירה ערך, ולא רשמנו בה אפילו פעם אחת את הפקודה **return**, נקבל שגיאת קומפילציה, והתוכנית לא תוכל להתחיל בריצה.

נתקן את הקוד שראינו מקודם, ע"י הוספת פקודת return מתאימה:



```
<global>
var result: number;
result = mulNum(2);
alert(result);

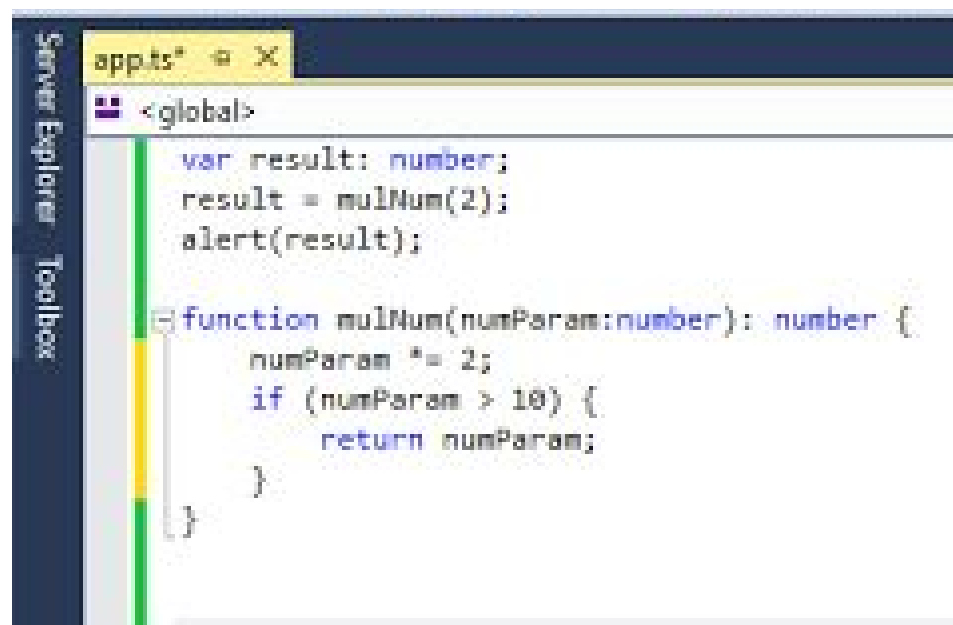
function mulNum(numParam:number): number {
    numParam*= 2;
    return numParam;
}
```

ואז, כאשר נריץ את התוכנית, נקבל את הפלט הבא:



נמשיך הלאה, ונניח שהפונקציה שעד עכשיו רק הייתה צריכה להחזיר את מכפלת המספר שנשלח אליה, השתדרגה, ותחזיר את מכפלת המספר רק במקרה בו תוצאת המכפלה תהיה גדולה מעשר.

לשם כך נשנה את הקוד, לקטע הבא:



```
app.ts
<global>
var result: number;
result = mulNum(2);
alert(result);

function mulNum(numParam:number): number {
    numParam *= 2;
    if (numParam > 10) {
        return numParam;
    }
}
```

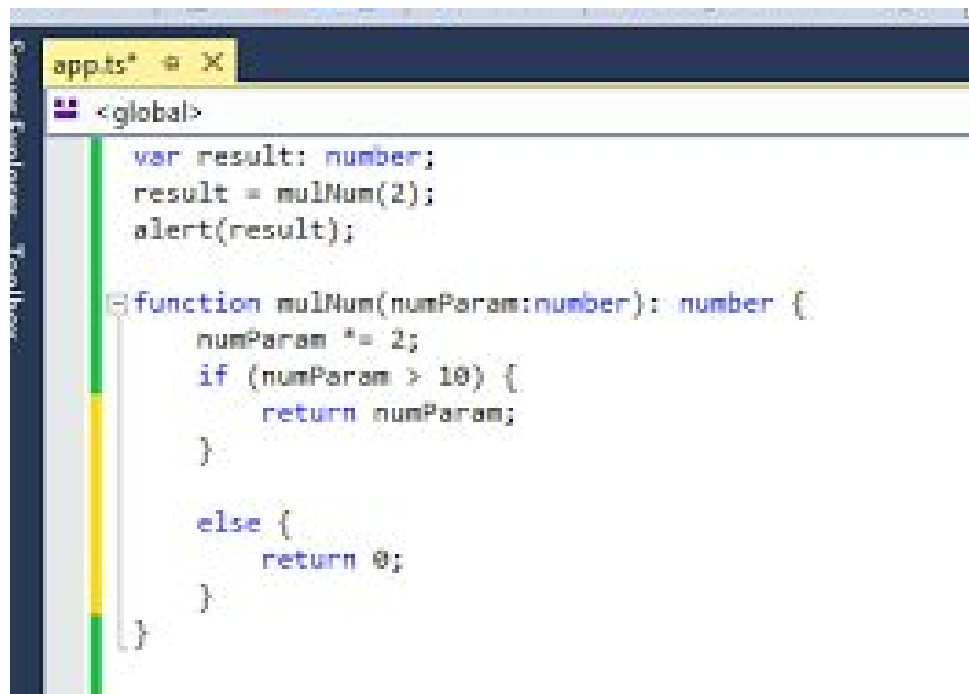
נריץ את התוכנית, ונקבל את התוצאה הבאה:



מיד נוכל לראות שקיימת בעיה בקוד שכתבנו, כיוון שהפלט שקיבלנו הוא undefined - שזהו פלט לא תקין. (כלומר - אנו רואים שהתכנית רצה ועברה קומפילציה ללא שגיאות קומפילציה, ובמהלך התוכנית וחישוב הביצועים היא הצליחה לרוץ ללא קריסה של שגיאת זמן ריצה, אבל הערך שהיא החזירה לנו הוא ערך שלא מתאים לחזור מפונקציה - ולכן ברור שיש כאן שגיאה לוגית)

מסקנה: כאשר יצרנו פונקציה שמחזירה ערך, ורשמנו בה לפחות פעם אחת את הפקודה return , התוכנית תוכל להתחיל בריצה, אבל אם עבור מצב מסויים לא יוחזר ערך - הדבר יחשב לשגיאה לוגית ויוחזר הערך undefined.

נתקן את הקוד שראינו מקודם, ע"י הוספת פקודת return מתאימה:



ואז, כאשר נריץ את התוכנית, נקבל את הפלט הבא:

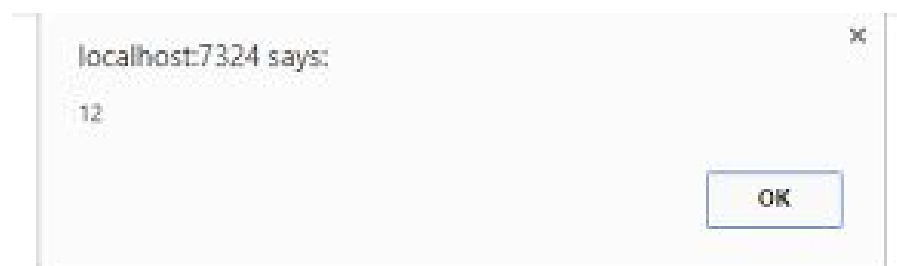


ואילו כאשר נשנה את הקוד, שבקריאה לפונקציה נשלח מספר שתוצאת מכפלתו גדולה מ 10:

```
Server Explorer - Toolbox
app.ts* X
<global>
var result: number;
result = mulNum(6);
alert(result);

function mulNum(numParam:number): number {
    numParam *= 2;
    if (numParam > 10) {
        return numParam;
    }
    else {
        return 0;
    }
}
```

אז, כאשר נריץ את התוכנית, נקבל את הפלט הבא:



דוגמאות לשגיאות אפשריות בפונקציה

שגיאת זמן ריצה

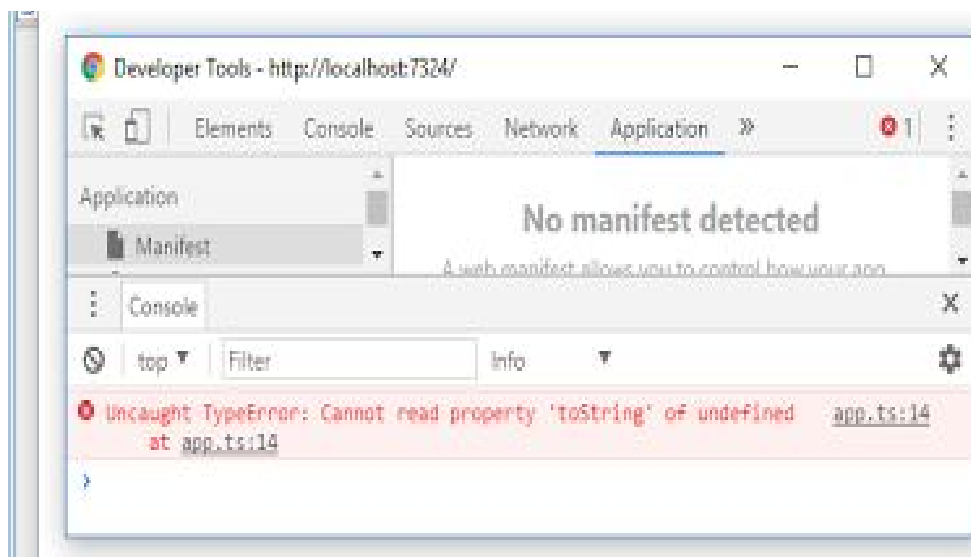
/////////////////MAIN SECTION/////////////////

var result: number;

```

var str: string;
var num: number;
str = "4";
num = Number(str);
result = mulNum(num);
//will cause a runtime error - because result is undefind
//and (undefind).toString() causes run time error
document.write(result.toString());
//////////FUNCTION SECTION//////////
function mulNum(x: number): number {
  if (x>10)
    return x*2;
}

```

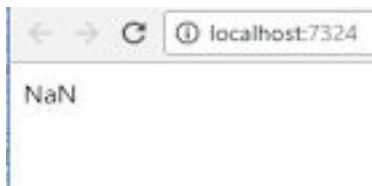


שגיאת לוגיקה

```

//////////MAIN SECTION//////////
var result: number;
var str: string;
var num: number;
str = "TypeScript";
num = Number(str);
result = mulNum(num); //---> num=NaN
document.write(result.toString());
//////////FUNCTION SECTION//////////
function mulNum(x: number): number {
  return x*2;
}

```



שגיאת קומפילציה

```
//////////////////MAIN SECTION//////////////////  
var result: number;  
var str: string;  
var num: number;  
str = "4";  
num = "4"; //compilation error - string cannot be assigned to a number  
result = mulNum(num);  
document.write(result.toString());  
//////////////////FUNCTION SECTION//////////////////  
function mulNum(x: number): number {  
    return x*2;  
}
```

