

# JavaScript Basic

# מה זה Javascript ?

זו שפת התכנות של HTML ושל האנטרנט

אין קשר בין Javascript לבין Java

Javascript הומצאה בשנת 1995 והפכה סטנדרט בשנת 1997

Javascript מתבסס **בצורה קריטית** על יצוג הדף כ DOM - Document Object Model

# משתנים - let

משתנים מוגדרים באופן מסורתי בעזרת let (בהמשך נראה שימוש גם בעזרת const, let) לדוגמא

```
let a,b=1,c;
```

```
a=2;
```

```
c=a+b;
```

שמות משתנים :

- שם משתנה מוגדר חד ערכית
- יש הבדל בין משתנה בשם x למשתנה בשם X זה נקרא case sensitive
- שמות משתנים חייבים להתחיל באות

נח לחשוב על משתנה כמו על קופסא עם שם וערך

# סוגי משתנים - data types

משתנים ב javascript יכולים להכיל מידע מסוגים שונים : בוליאני, אובייקטים, מחרוזות, מספרים, undefined, null אבל את **כולם** אפשר להגדיר בעזרת let

let num = 1.5, name='123', person = {firstName:'John', lastName:'Deer'}, d=true, e;

d = name + num;

מותר ב JS, לצורך הפעולה num יחשב כמחרוזת ולכן גם d יהיה מחרוזת

let x = 10 + 4 + "V1";

y לא שווה x כי החישוב מתבצע משמאל לימין

let y = "V1" + 10 + 4;

let s1='cat', s2="tiger";

ניתן להגדיר מחרוזת באמצעות גרש אחד או גרש כפול

alert(e);

יציג undefined שזה ערך משתנה שלא קבל ערך

# ביטויים (expressions) ומשפטים (statements)

משפט : שורת קוד - מסתיימת בנקודה פסיק.(לא תמיד חובה אבל רצוי) לדוגמה

```
x = x + 1; console.log("hello java script");
```

ביטוי : שילוב של אחד או יותר מספרים , משתנים , אופרטורים ופונקציות אשר יוצרים ערך.

דוגמאות :

- $x+1$
- $x=x+1$
- $x > 1$
- 1

# אופרטורים חשובים - operators

פירוט	משפחה
+	אריتمטיים
-	
*	
/	
%	
++	
--	
=	השמה
+=	
-=	
*=	
/=	
%=	
==	השוואה
===	
!=	
!==	
<	
>	
<=	
>=	
?	
&&	לוגיים
!	

האם ערך שווה  
וגם סוג

האם ערך לא שווה  
או סוג לא שווה

Conditional operator

דוגמה

let a=1,b="abc",c;

c=1+b;// זה מותר

# קלט פשוט מהמשתמש

```
alert('password is not correct');
```

```
let confirmResult = confirm('do you want to proceed to checkout');
```

מחזיר ערך בוליאני

```
let promptResult = prompt('please insert rectangle width');
```

מחזיר את הערך שהוכנס  
**כמחרוזת** !!!! או null אם לחצו על  
הכפתור cancel. במידה והערך  
הוא מספר רצוי להמירו בעזרת  
Number

# אופרטורים - אופרטור השמה

לדוגמא השמת ערך 7 בתא המיצג על ידי המשתנה sum.

```
sum = 7;
```

**שימו לב !** הפעולה מתבצעת מימין לשמאל ז"א קח את הערך בימין קרי 0 ושים אותו בתא משמאל קרי sum.

אפשר גם

```
min = max = 0;
```

קודם מתבצעת השמה של 0 לתוך max (שעכשיו ערכו 0) ואחר כך מתבצעת השמה של הערך בmax לתוך min.

האם מותר לרשום ?

```
sum + 1 = 2 ;
```

<http://www.nathankrasney.com/>



# אופרטורים - אופרטורים מתמטיים +, -, \*, /

let width=9,height=23, area ,ratio;

width = width + 1; // ערך חדש 10

height = height -3; // ערך חדש 20

area = width \* height; // ערך חדש 200

ratio = height / width; // ערך חדש 2

# אופרטורים - אופרטורים מתמטיים %

האופרטור % נקרא אופרטור מודולו והוא אופרטור שארית

ערך חדש 2 // 3 % 20 = result

# אופרטורים - אופרטורים מתמטיים מורכבים

אופרטור	תיאור	שווה ערך לא מורכב
+=	$x += 4$	$x = x + 4$
-=	$x -= 4$	$x = x - 4$
*=	$x *= 4$	$x = x * 4$
/=	$x /= 4$	$x = x / 4$
%=	$x \% = 4$	$x = x \% 4$

# אופרטורים - אופרטורים מתמטיים הגדלה ב1 והקטנה ב1

ניתן להשתמש באופרטור ++ להגדלה ב1 ובאופרטור -- להקטנה ב1.

```
let a=3,b;
```

```
a++; //4 ערך חדש
```

```
a=a+6; //10 ערך חדש
```

```
a--; //9 ערך חדש
```

```
--a; //8 ערך חדש
```

שימו לב : בהשמה יש חשיבות אם האופרטור בצד ימין או שמאל

זה לא היה נותן אותה תוצאה אם ++ היה מימין // b = ++a;

# אופרטורים - אופרטורים של יחס

אופרטור	תיאור
>	גדול מ
<	קטן מ
==	שווה ל
!=	לא שווה ל
>=	גדול מ או שווה ל
<=	קטן מ או שווה ל

# אופרטורים - דוגמאות אופרטורים של יחס

```
let width=3.4, height=4.0;
```

```
let count=10;
```

תוצאה	ביטוי
false	width > height
true	width < 5
true	count == 10
true	count != 11
true	count >= 10
false	count <= 9

# קדימויות - Operator precedence

```
let now = 2019;  
let sarit = 1961;  
let fullAge = 18;  
let isSaritFullAge = now - sarit > fullAge;  
console.log(isSaritFullAge);  
isSaritFullAge = (now - sarit) > fullAge;  
console.log(isSaritFullAge);
```

זה עובד כי אופרטור - נמצא בעדיפות גבוהה אופרטור > בעדיפות יותר נמוכה ואופרטור = בעדיפות עוד יותר נמוכה.

אופרטור סוגריים הוא בעדיפות הכי גבוהה ולכן **מומלץ במקרה של ספק לשים סוגריים**

הטבלה המלאה מופיעה [כאן](#) :

- Precedence קובע עדיפות, בעל עדיפות גבוהה יבוצע לפני מי שיש לו עדיפות נמוכה
- Associativity קובע את הסדר כאשר האופרטורים בעלי אותה עדיפות

## דוגמא קדימויות

```
let age1 = 18 , age2=22 , age3 = 14;
```

```
let average = age1+age2+age3/3;
```

```
console.log(average);
```

האם החישוב נכון

A light gray rectangular callout box with a black border and a pointed bottom. A black arrow points from the left side of the box to the division part of the code 'age3/3' in the line 'let average = age1+age2+age3/3;'. The text inside the box is 'האם החישוב נכון'.



# Debugger

מומלץ להשתמש ב debugger במקרה שהקוד לא מבצע את מה שאנו מצפים

נכנסים ל chrome dev tools ואז לטאב של sources שמים breakpoint ועושים refresh לדף

מושגים חשובים :

- Breakpoint
- Watch
- Step over
- Step into
- Resume
- Call stack , Scope , Global

# מבני בקרה

משפטי בקרה מורכבים מתנאים ולולאות ומאפשרים לנו לשלוט על זרימת התכנית כפי שראינו בתרשימי זרימה.

תנאים מיצגים בעזרת switch , if else , if (אלה מילים שמורות בשפה - keyword)

לולאות מיצגים בעזרת do while , while , for (מילים שמורות)

# המרה בין סוגים - Data types conversion

```
let num1String = "3";
```

```
let num2String = "4";
```

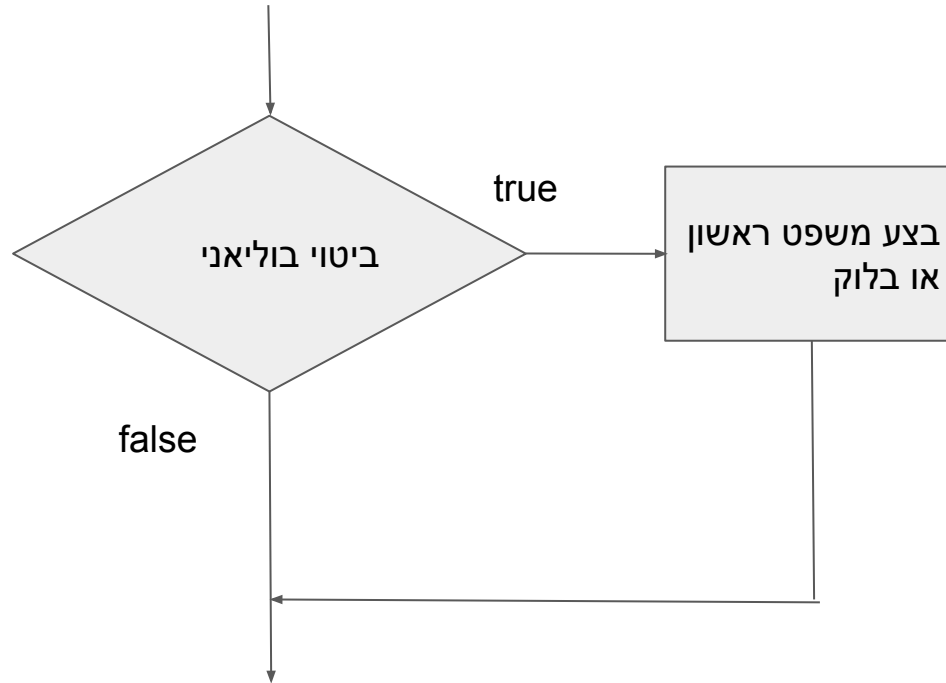
```
let result = Number(num1String)+Number(num2String);
```

```
console.log(`result : ${result}`);
```

מה היתה התוצאה - result אם  
לא היינו משתמשים בהמרה  
בעזרת Number ?

# מבני בקרה - תנאי if כתאור גרפי

תנאי if ניתן לתיאור גרפי לדוגמא בצורה :



# מבני בקרה - משפט / בלוק

מהו משפט ב JS :

משפט הוא ביטוי בשפת JS שמסתיים בנקודה פסיק לדוגמא

```
b = a+b;
```

מהו בלוק ב JS :

בלוק הוא משפט או אוסף משפטים אשר מוגדרים באמצעות סוגרים מסולסלות לדוגמא

```
{
```

```
b=a+1;
```

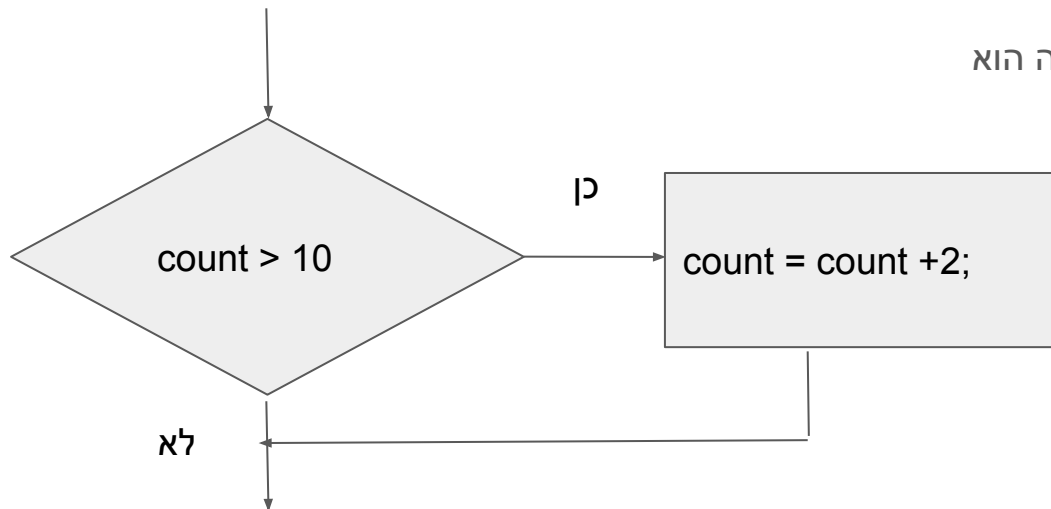
```
a=a+2;
```

```
}
```

# מבני בקרה - if עם משפט אחד לביצוע

נניח שיש לנו מונה בשם count ואנו רוצים לקדם את ערכו ב2 אם ערכו גדול מ10.

תרשים הזרימה הוא



והקוד הוא

```
if(count > 10)
```

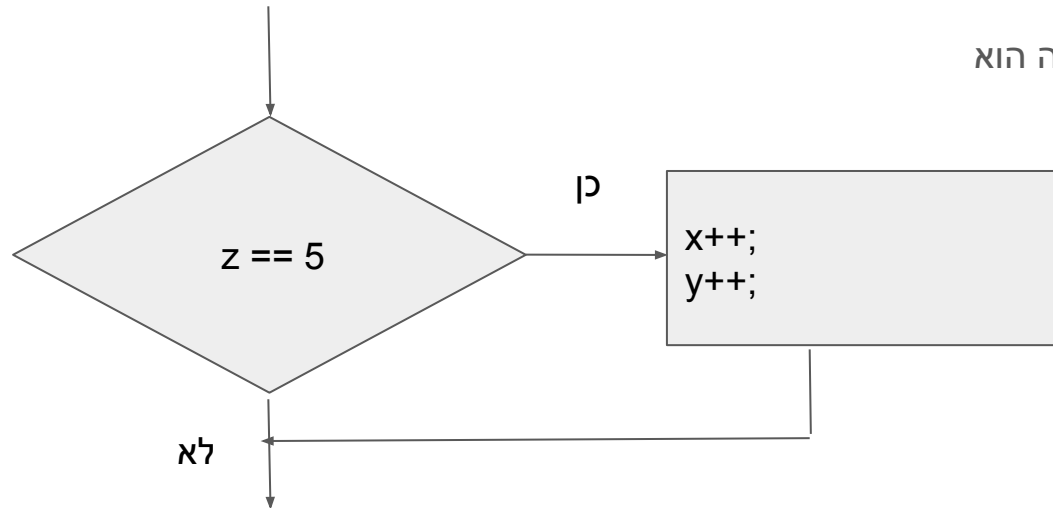
```
count=count+2;
```

הערה : בדוק בתכנית לחישוב מספר מקסימלי האם יש שימוש ב if

# מבני בקרה - if עם בלוק

נניח שיש לנו שלושה משתנים x,y,z ורוצים לקדם את x,y ב 1 אם z שווה 5

תרשים הזרימה הוא



והקוד הוא

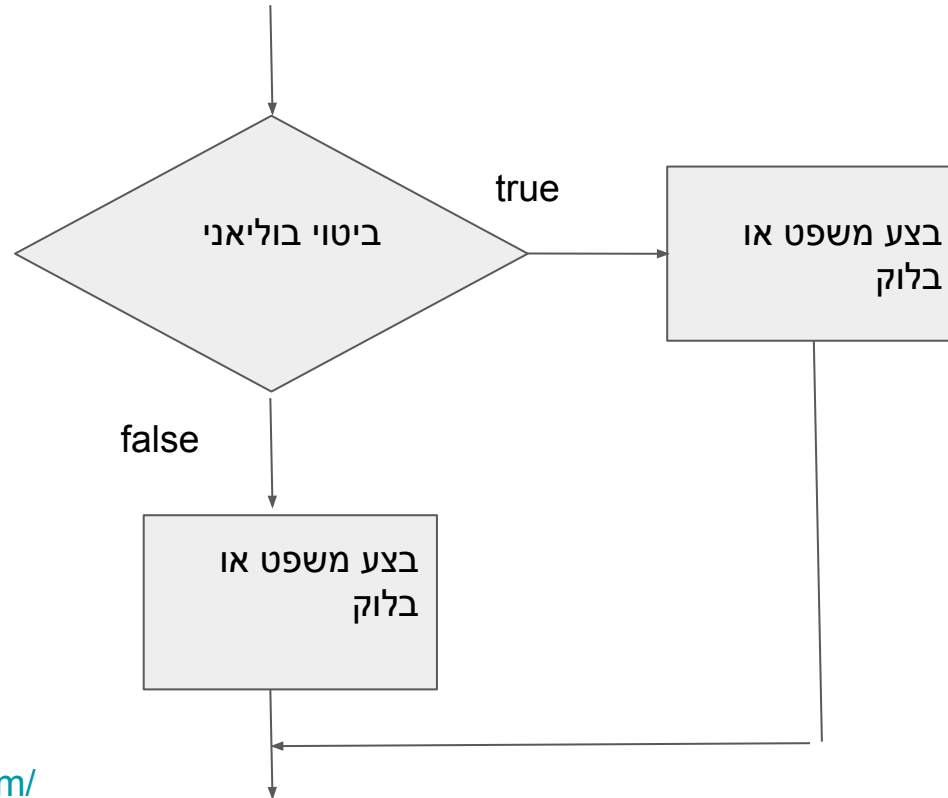
```
if(z == 5){
```

```
  x++;
```

```
  y++;};
```

# מבני בקרה - תנאי if else כתאור גרפי

תנאי if else ניתן לתיאור גרפי לדוגמא בצורה :

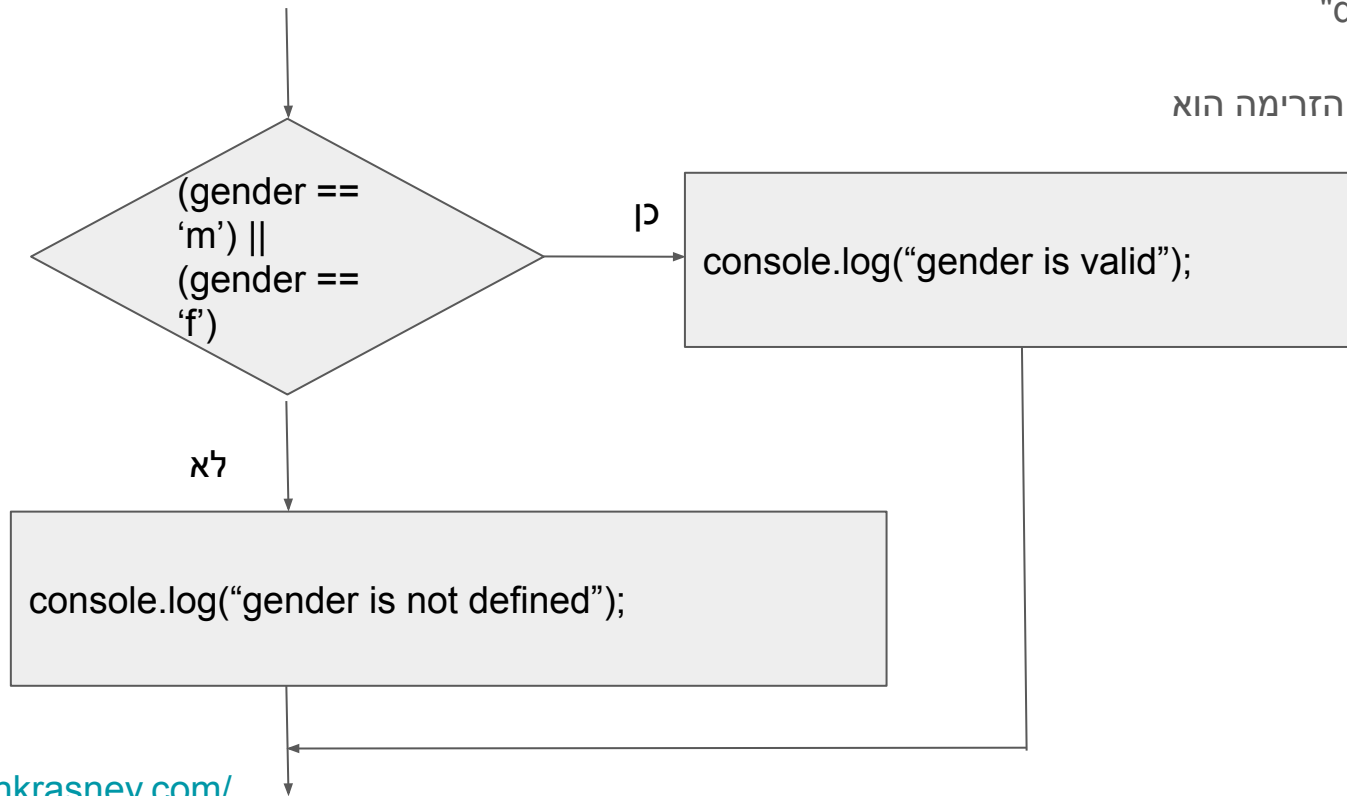




# מבני בקרה - if else בתרשים זרימה

נבדוק שיש לנו משתנה מסוג char בשם gender. אם ערכו 'f' או 'm' נדפיס למסך "gender is valid" אחרת "gender not defined"

תרשים הזרימה הוא



# מבני בקרה - if else בקוד

```
if((gender == 'm') || (gender == 'f'))
```

```
  console.log("gender is valid");  
else
```

```
  console.log("gender is valid");
```

הערה :  
למרות שאין הכרח מומלץ **תמיד** להכניס את המשפט לביצוע לסוגרים מסולסלים

```
if((gender == 'm') || (gender == 'f')){
```

```
  console.log("gender is valid");  
}  
else{
```

```
  console.log("gender is valid");  
}
```

# מבני בקרה - if else מרובים

נתון משתנה - תו בשם gender. אם ערכו 'm' נדפיס 'gender is male' אם ערכו 'f' נדפיס 'gender is female' אחרת "gender not defined"

```
if(gender == 'm'){
```

```
  console.log("gender is male");  
}
```

```
else if (gender == 'f'){
```

```
  console.log("gender is female");  
}
```

```
else{
```

```
  console.log("gender is not defined");  
}
```

# מבני בקרה - switch

לעיתים נוח יותר להשתמש בswitch כשיש ערכים שונים **לאותו משתנה** . עבור הדוגמה מהשקף האחרון

```
switch(gender){
```

```
case 'm':
```

```
console.log("gender is male");
```

```
break;
```

break מוציא מידית  
מהבלוק

```
case 'f':
```

```
console.log("gender is female");
```

```
break;
```

יתבצע רק אם שום  
break לא התבצע

```
default:
```

```
console.log("gender is not defined");
```

```
}
```

- מותר גם כמה case מעל אותו break
- דוגמא מעולה בקבלת שם חודש בשימוש עם Date

# Conditional operator - ternary

```
let message, age=45;
```

```
message = age < 18 ? "You are young ,you can not enter the club" : "Welcome to  
our club";
```

# Truthy and Falsy

**truthy** הוא ביטוי - expression אשר נחשב לערך בוליאני true כאשר הוא נמצא בהקשר בוליאני לדוגמא תנאי

**falsy** הוא ביטוי - expression אשר נחשב לערך בוליאני false כאשר הוא נמצא בהקשר בוליאני לדוגמא תנאי

כל הביטויים הם **truthy מלבד**

**נחשבים כ falsy**

false , 0, "",",null,undefined,NaN

# מבני בקרה - לולאות

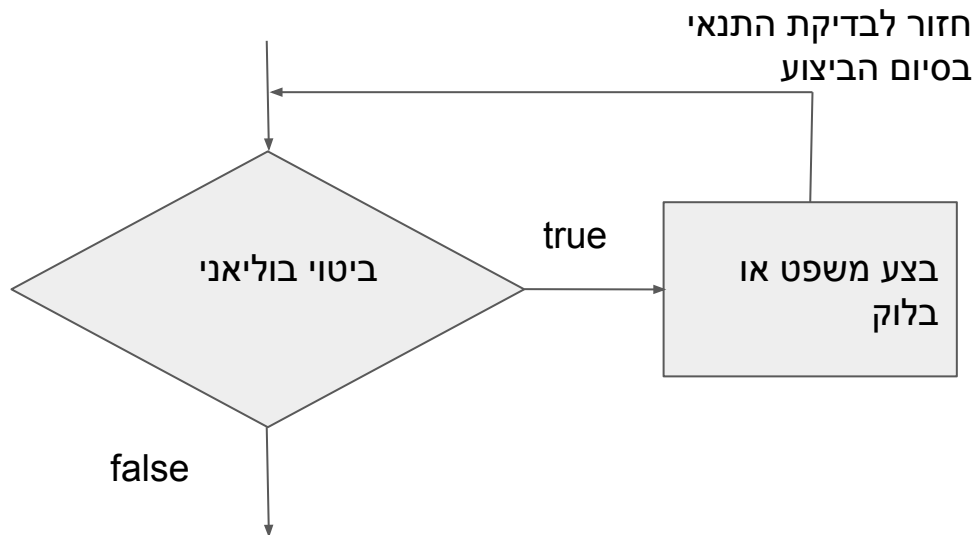
ישנם מקרים רבים בהם אנו נדרשים לחזור על פעולה מספר פעמים.

הפעולה הזו נקראת בתוכנה לולאה.

תרשים זרימה של לולאות נראה מאוד מאוד דומה לתרשים של if ובהבדל אחד - לאחר ביצוע המשפט או בלוק חוזרים לבדיקת הערך הבוליאני (נראה בשקף הבא)

# מבני בקרה - תרשים זרימה while

תרשים זרימה של while נראה מאוד מאוד דומה לתרשים של if עם הבדל אחד - לאחר ביצוע המשפט או בלוק חוזרים לבדיקת הערך הבוליאני בצורה הבאה :





# מבני בקרה - פקודת while

פקודת while מאפשרת ביצוע לולאה ונרשמת בצורה הבאה :

while (ביטוי בוליאני)

אם true אז מבצע משפט או בלוק

בצע משפט או בלוק

לדוגמה : נתון מספר שלם number בצע סיכום של המספרים השלמים 1 עד number

```
let number = 3, sum=0;
```

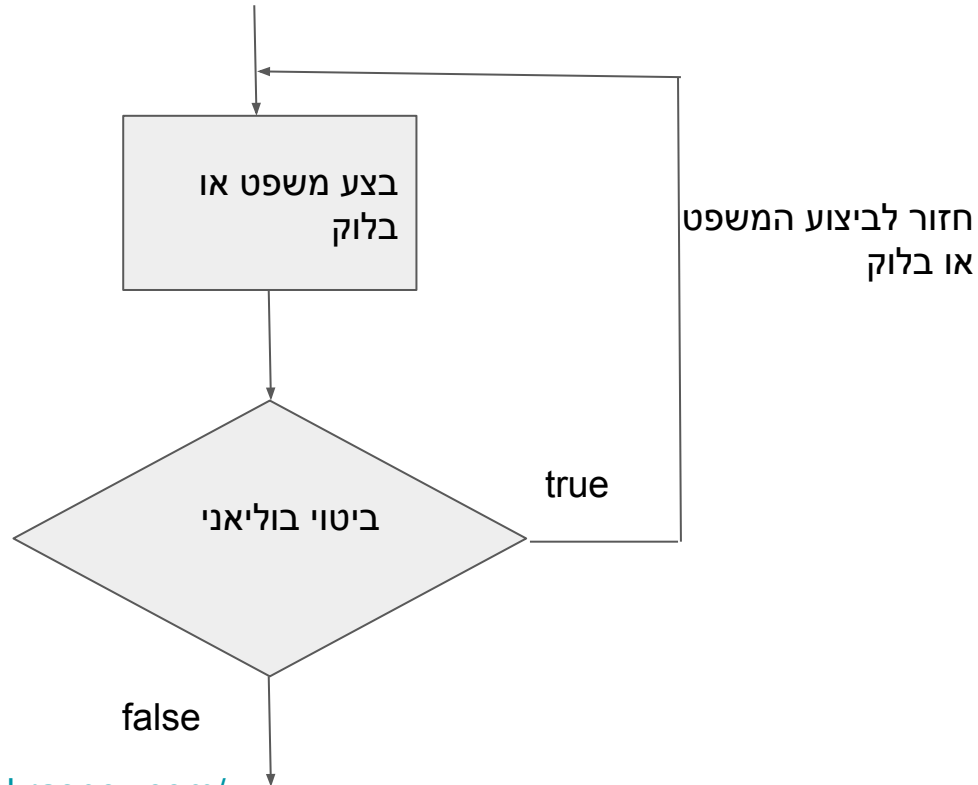
```
while (number > 0){
```

```
    sum = sum + number;
```

```
    number--;
```

```
}
```

# מבני בקרה - תרשים זרימה do while



# מבני בקרה - פקודת do while


פקודת do while מאפשרת ביצוע לולאה ונרשמת בצורה הבאה :

do

בצע משפט או בלוק

while (ביטוי בוליאני)

אם true אז מבצע  
משפט או בלוק



לדוגמה : נתון מספר שלם number בצע סיכום של המספרים השלמים 1 עד number

```
do {
```

```
    sum = sum + number;
```

```
    number--;
```

```
} while (number > 0);
```

# מבני בקרה - פקודת for

אם true אז מבצע  
משפט או בלוק

**כל** לולאה אפשר לעשות עם while אולם לעיתים נוח יותר להשתמש ב for

פקודת for היא בעלת המבנה הבא

(משפט לביצוע בסוף הלולאה ; ביטוי בוליאני ; אתחול) for

בצע משפט או בלוק

כל לולאת for אפשר להביא לצורת while הבאה :

אתחול;

while(ביטוי בוליאני)

בצע משפט או בלוק

;משפט לביצוע בסוף הלולאה

אחרי שנלמד עם  
מערכים נכיר את  
foreach

# מבני בקרה - פקודת for דוגמא

לדוגמה : נתון מספר שלם number בצע סיכום של המספרים השלמים 1 עד number

```
let number = 3, sum=0;
```

```
for(let i= 1 ; i <= number ; i++)
```

```
{
```

```
    sum = sum + i;
```

```
}
```

# מבני בקרה - continue , break

לעיתים נדרש לצאת מלולאה לפני שהתנאי לעצירה התקיים - ניתן לעשות זאת באמצעות break

לעיתים נרצה לקפוץ לסוף הלולאה הנוכחית - ניתן לעשות זאת בעזרת continue

# מבני בקרה - לולאה מקוננת

לולאה מקוננת היא לולאה אשר מכילה לולאה נוספת. לדוגמה כתוב תכנית אשר מדפיסה את לוח הכפל :

```
let mul;

for (let row = 1; row <= 10; row++){

  for (let col = 1; col <= 10; col++){

    mul = row * col;

    console.log(`${row} * ${col} = ${mul}`);

  }

}
```

# פונקציות - הגדרה + דוגמה

מהי פונקציה : אוסף של משפטים בשפת JavaScript אשר יחדיו מבצעים משימה.

פונקציה עוזרת לנו לרכז במקום אחד משימה קצרה שעשויה לחזור על עצמה , לדוגמה חישוב שטח מלבן

מגדירים פונקציה באמצעות :

- שם הפונקציה
- רשימת משתנים בכניסה לפונקציה (ארגומנטים/פרמטרים)
- גוף הפונקציה

לדוגמה פונקציה שמחשבת שטח של מלבן

```
function computeArea( nWidth, nHeight){  
  
return nWidth*nHeight;// מחזיר ערך  
  
}
```

שם הפונקציה : ComputeArea  
רשימת משתנים : nWidth, nHeight  
גוף הפונקציה : ;return nWidth\*nHeight



# מקרים שונים בפונקציות

פונקציות יכולות להיות מהסוג הבאים **ושילוב** שלהם

1. ארגומנטים לפונקציה : כן / לא
2. הארגומנטים לפונקציה יכולים להיות משתנים כן / לא (לדוגמא מספר)
3. שימוש במשתנים לוקלים כן / לא
4. החזרת ערך עם return כן / לא
5. שימוש במשתנים מ scope חיצוני כן / לא
6. משתנה לוקלי או ארגומנט עם שם זהה למשתנה ב scope חיצוני

# פונקציה - הגדרה וקריאה

פונקציה מוגדרת על ידי אוסף של משפטים שנועדו לעשות משימה מסוימת.

הפונקציה תבוצע כאשר יקראו לה.

```
function sum(a, b) {  
    return a + b;  
}
```

```
alert(sum(2, 3));
```

המשתנה a יקבל את הערך 2  
והמשתנה b יקבל את הערך 3

הערה : לפונקציה יכולים להיות גם  
ערכי ברירת מחדל - default

# תחום מחיה של משתנים - letiable scope

Scope הוא אוסף המשתנים אליהם יש גישה.

דוגמה	הסבר	סוג משתנה
<pre>function f() {   let a = 1;   //..... }</pre>	משתנים שמוגדרים בתוך פונקציה ניתנים לגישה מתוך הפונקציה בלבד	מקומי - <b>local</b>
<pre>let a = 1; function f1() { a++;} f1(); alert(a); // --- show 2</pre>	משתנה שמוגדר מחוץ לפונקציה נחשב גלובלי וניתן לגישה גם מחוץ לפונקציה וגם בתוכה. המשתנים האלה שייכים לאובייקט window ואפשר לגשת אליהם גם דרכו	גלובלי - <b>global</b>
<pre>function f2() {a1 = 10;   a1++;} f2(); alert(a1);// --- show 11</pre>	<b>זהירות</b> : משתנה שלא מוגדר בתוך הפונקציה הופך להיות גלובלי באופן אוטומטי	גלובלי אוטומטי

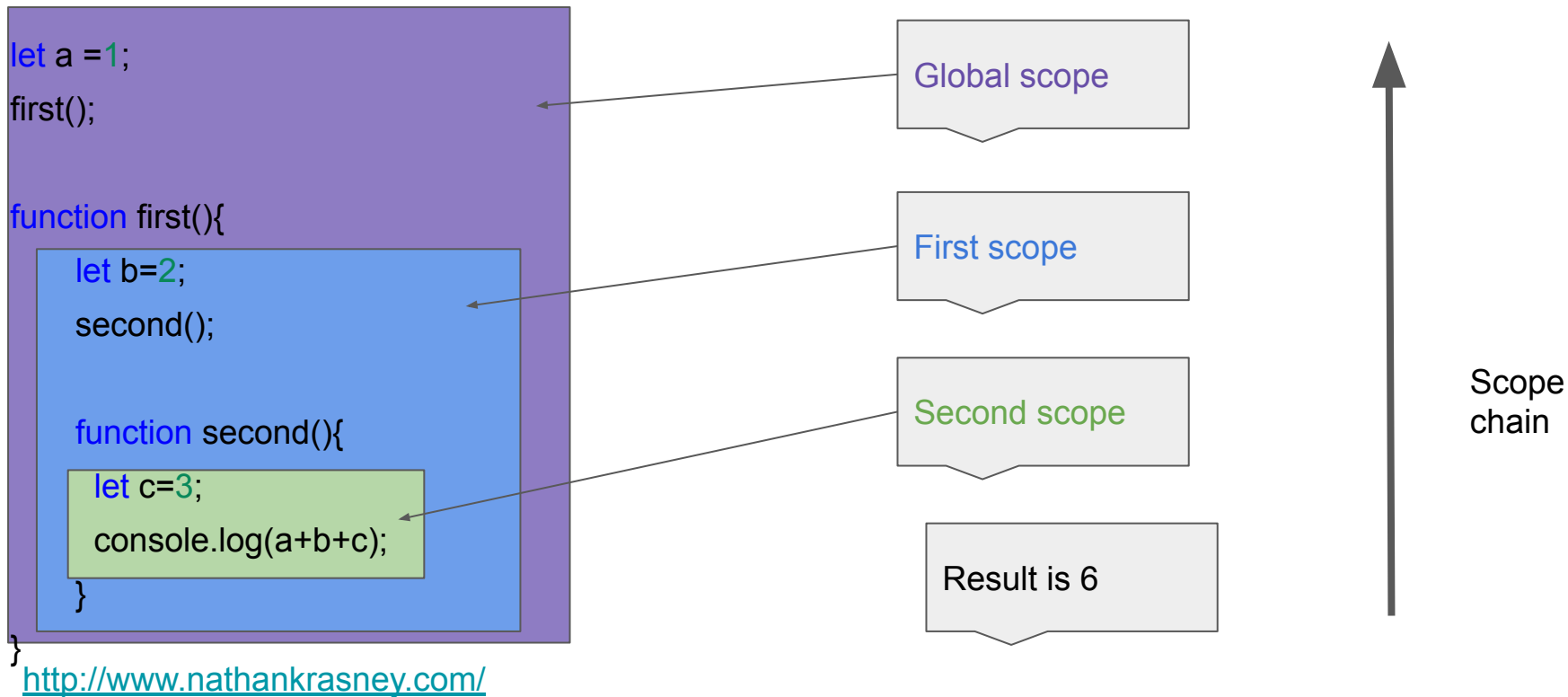
# פונקציה - הרחבות

Arrow function

Anonymous function

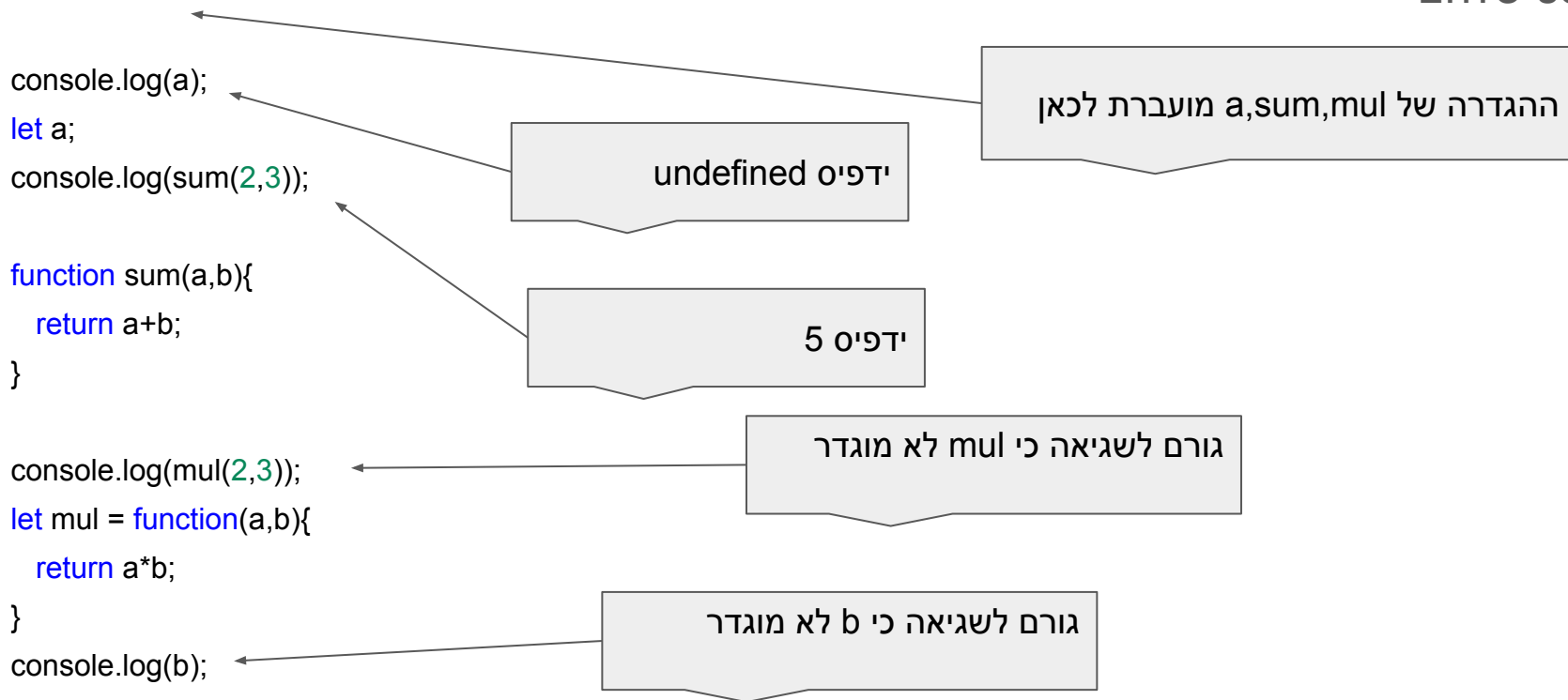
# lexical scope

A function that is lexically inside another function get access to the scope of the outer function



# Hoisting

Hoisting הוא מכניזם ב javascript בו הגדרת פונקציות ומשתנים מועלים לפני הרצה לתחילת ה scope שלהם



# מערך - מהו מערך

מערך הוא אוסף של ערכים (ב C# הם מאותו סוג אבל ב JS זה יכול להיות מסוגים שונים)

מערך הוא מבנה נתונים (ב C# מספר האיברים קבוע אבל ב JS מספר האיברים הוא דינמי)

דוגמאות :

- מערך לשמירת כל האנשים במשרד הפנים
- מערך לשמירת מספר הטלפון של החברים שלי

אפשר לחשוב על מערך כמו ארון עם מגירות : מגירה ראשונה , מגירה שניה , מגירה שלישית ....  
ובכל מגירה ניתן לשים ערך. על מנת לגשת לכל מגירה משתמשים במספר של המגירה - אינדקס.

למגירה הראשונה ניגשים עם אינדקס 0 , למגירה השנייה עם אינדקס 1 וכך הלאה

# מערך - יצירת מערך חד מימדי

יוצרים מערך בעזרת אופרטור [].

לדוגמא יצירת מערך של איברים בשם Numbers בעל 5 איברים :

```
let Numbers = [3,5,8,9,11];
```

הצגה הזיכרון של Numbers היא :





# מערך - השמת ערכים במערך חד מימדי

הגישה ז"א כתיבה - השמה וקריאה לכל תא ("מגירה") במערך נעשית בעזרת []

```
let Numbers = [];
```

```
Numbers[1] = 5;
```

```
Numbers[2] = 7;
```

```
Numbers[3] = 9;
```

```
Numbers[4] = 11;
```

```
Numbers[0] = 3;
```

כעת תמונת הזיכרון של Numbers היא



3	5	7	9	11
---	---	---	---	----

# מערך - קריאת ערכים במערך חד מימדי

כתיבה וקריאה לכל תא ("מגירה") במערך נעשית בעזרת אופרטור []

```
let sum = 0;
```

```
console.log(`Numbers[2] : ${Numbers[2]}`);
```

```
for (let i = 0; i < 5; i++){
```

```
    console.log(`Numbers[${i}] : ${Numbers[i]}`);
```

```
    sum += Numbers[i];
```

```
}
```

```
console.log(`sum : ${sum}`);
```

# פעולות על מערך

יש [המון](#) פונקציות אבל החשובות הן :

push , splice , find , filter , map , reduce

מאפיין חשוב

length

מעבר על איברים במערך

```
for (person in presons){...}
```

# מערך - אובייקט

מערך נחשב אובייקט (נבין בפרק הבא טוב יותר מהו אובייקט)

אובייקט בכלל ומערך בפרט נחשבים כ reference type (נבין בפרק הבא טוב יותר למה הכוונה)

# אוסף של תווים / string

אפשר לייצג אוסף של תווים בעזרת מערך לדוגמא

```
let a = [ 'h', 'e', 'l', 'l', 'o' ];
```

בד"כ נוח יותר לייצג אוסף של תווים בעזרת מחלקה (נרחיב בפרק 2) שנקראת string לדוגמא

```
let s="123";
```

```
for (let i = 0; i < s.length; i++){
```

```
  console.log(s[i]); }
```

תכונה מופיעה ללא סוגריים - Property

מופיעה עם סוגריים עגולם פונקציה - Method

אבל string הוא **immutable** ולכן לא ניתן לבצע

```
s[0] = '4'; // הפעולה לא תתבצע
```

# פעולות על מחרוזת - string

length - מאפיין שמחזיר את מספר התווים במחרוזת

substr - פונקציה שמחזירה חלק מהמחרוזת בהינתן אינדקס התחלה ומספר תווים

split - פונקציה שמחזירה מערך בהנתן separator

indexOf - פונקציה שמחפשת מחרוזת ומחזירה אינדקס שלו או -1

Number - פונקציה שמחזירה מספר בהינתן מחרוזת

# מערך - מערך דו מימדי

הגדרה + אתחול :

```
let array2D = [  
    [0, 1, 2, 3] ,  
    [4, 5, 6, 7] ,  
    [8, 9, 10, 11]  
];
```

# מעריך - מעריך דו מימדי

גישה לאברי המעריך הדו מימדי array2D בשקף הקודם

גישה לאיבר השלישי בשורה השניה :

```
array2D[1][2]
```

הדפסה ל console של כל האיברים במעריך

```
for (let i = 0; i < 3; i++)
```

```
for (let j = 0; j < 4; j++){
```

```
    console.log(`${array2D[i][j]}`);
```

```
}
```



# Debugging

אין בjavascript קומפילר ולכן אפשר להעלות דף HTML עם שגיאות תחביר.

לעתים אין שגיאות תחביר אבל יש שגיאה לוגית אותה נדרש למצוא. להלן אופציות ל debugging

הסבר	אופצית debugging
ראינו בשקפים הקודמים שאפשר לכתוב למסך. בהקשר הזה אפשר להשתמש console.log (ללחוץ על F12) או alert	output
לדוגמא ל chrome יש כלי שמאפשר לשים breakpoint ולראות משתנים. יש לפתוח את דף הHTML דרך chrome אחר כך F12 אחר כך sources, לוחצים על השורה מסוימת בקוד מצד שמאל להכנסת breakpoint, מבצעים רענון לדף וניתן לעשות watch, step by step באמצעות התפריט מימין	כלים של דפדפנים

# שימוש ב Javascript מקובץ חיצוני

אפשר לשים קוד javascript בקובץ חיצוני ואז להפנות אליו מקבץ html דרך אלמנט script בצורה הבאה

```
<body>
```

```
<script src="someJavascriptFile.js"></script>
```

```
</body>
```

אפשר לשים את השורה הזו או ב head או ב body. רצוי לשים אותה בסוף body משיקולי תצוגה מהירה של הדף כי כך קודם מוצג הדף ורק אחר כך מועלה קובץ javascript

בקובץ javascript החיצוני רושמים את הקוד לדוגמא

```
alert('hello world');
```

# הערות - comments

אפשר להגדיר בשתי דרכים

```
// this is a comment for one line
```

```
/* this is
```

```
comment for
```

```
few lines
```

```
*/
```

# console tricks

שם	תיאור	דוגמא
clear()	מנקה כל מה שכתוב	console.clear()
table(...)	יוצר טבלה (דורש עבודה עם אובייקטים)	<pre>let name1 = { firstName : "Nathan", lastName : "Krasney" } let name2 = { firstName : "Sarit", lastName : "Agry" } console.table([name1, name2]);</pre>
%c	עיצוב	console.log('%c 456','color:red; font-size:50px;');
time(..), timeEnd(..)	מציג זמן ביצוע קוד	<pre>console.time('tag1') console.log('do something') console.timeEnd('tag1')</pre>

# מקורות

**W3schools.com** - מצוין בשביל להבין מושגים, מכיל דוגמאות להרצה ושינוי והסברים קצרים, ישר ולעניין

**Developer.mozilla.org** - מצוין בשביל להרחיב מעבר ל w3schools

שני האתרים מקצועיים וניתן לסמוך עליהם (לא רק בJavaScript)

וידאו מעניין על ההיסטוריה של JavaScript [כאן](#) וסרטון מתקדם ומעניין [כאן](#)