**מדריך JavaScript בעברית | אסף וליקט אורי ברעם**

[**https://ori-baram.dev**](https://ori-baram.dev/)

**רצוי מאוד לכתוב קוד JavaScript במצב strict mode:**

**שימוש:** בתחילת קובץ JS פשוט כותבים 'use strict'.

**יתרונות**

* מונע יצירת באגים ושגיאות בהמשך.
* מגביל אותנו מלכתוב קוד "חלש ופגיע"
* יוצר שגיאות בולטות לעין בקונסול במקרים מסוימים, שבלי מצב ה-strict mode הקוד היה קורס ולא היינו יודעים למה.
* מונע שימוש בשמות משתנים שהם מילים שמורות לשימוש עתידי ב-JavaScript כמו private ו-interface לדוגמה

**פרמטר** – מה שמוגדר שהפונקציה מקבלת במעמד כתיבת/הגדרת הפונקציה

const foo = (num) => console.log(num)

**ארגומנט** – מה שמעבירים לפונקציה בפועל במעמד הרצת הפונקציה

foo(2)

**Type coercion**

'string' + num = string

num + 'string' = string

'string' – num = num

'string' / 'string' = num

'string' > 'string' = true || false

'30' > '8' = true // === 30 > 8

**אופרטור typof**

שימוש – "typeof something"

מחזיר את סוג המשתנה – string, number, Boolean...

**6 falsy values**

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

**Statement VS Expressions**

**Statements**:

3+4

1991

value || value

value && value

**expressions**:

if{…}

**טיפ לעיצוב באמצעות JavaScript**

עדיף לעבוד עם קלאסים ולא ישירות עם הסטייל של האלמנטים

~~element.style.property = new style~~

element.classList.add('new-style')

element.classList.remove('new-style')

**מערכים**

**Array Methods**

**.slice**

לא משנה מערך קיים – מחזירה מערך חדש.

.slice(x) – מחזירה העתק של המערך מהנקודה המצויינת עד לסוף המערך.

.slice(x, y) – מחזירה העתק מהנקודה עד לאינדקס שצויין לא כולל.

.slice(-x) – מחזירה העתק מהסוף.

.slice(0, -2) – תחזיר הכל חוץ משני האחרונים.

.slice() – בלי ארגומנטים תחזיר **shallow copy** של המערך (כלומר אם יש במערך אובייקטים ומערכים למשל, הם לא יועתקו – המערך החדש יפנה לאותו המקום שלהם בזיכרון ושינוי שלהם במערך החדש ישנה אותם גם במערך הישן).

**.splice**

משנה מערך קיים – מחזירה את מה שנלקח מהמערך המקורי.

אותם פעולות כמו slice אבל משנה את המערך.

.splice(x, y) – x נקודת ההתחלה, y מספר האלמנטים שרוצים למחוק החל (וכולל) מהאינדקס שצויין.

**.reverse**

(הופכת) משנה את המערך הקיים ומחזירה אותו הפוך (לא יוצרת עותק).

**.concat**

* ממזג מערכים

arr1 = [1, 2, 3];

arr2 = [4, 5, 6];

arr3 = arr1.concat(arr2) // [1, 2, 3, 4, 5, 6]

**\* כיום, עם ES6, במקום concat עדיף להשתמש ב-spread operator – […arr1, …arr2]**

**.join**

.join(', ') – מחזירה **סטרינג** כאשר מה שהוגדר בסוגריים מפריד בין האלמנטים מהמערך המקורי.

**.findIndex**

arr.findIndex(el => el === condition)

מחזירה את האינדקס הראשון במערך שעונה על התנאי.

**.some**

arr.some(el => el > 1)

מחזירה ערך בוליאני על בסיס התנאי שניתן – בודקת אם קיים לפחות אלמנט אחד שעונה על התנאי (בניגוד למתודת includes שמחפשת שוויון מוחלט, כאן נבדק קיומו של תנאי).

**.every**

מחזירה בוליאני כאשר כל האלמנטים עונים על התנאי

arr.every(el => el > 0)

**Math.abs(-7)** – הפיכת ערך שלילי לערך חיובי

**.flat**

משטחת מערך ברמה ראשונה:

const arr1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];

const arr2 = arr1.flat() // [1, 2, 3, 4, 5, 6, 7, 8, 9]

לא משטחת מערכים בתוך מערכים – כדי שכן תשטח מערכים בתוך מערכים מעבירים למתודה את עומק הרמות שרוצים לשטח – arr.flat(2)

**.flatMap** – שילוב של מתודת map יחד עם מתודת flat – היא קודם ממפה ואז עושה שיטוח – אבל לא ניתן להגדיר לה עומק שיטוח. היא יכולה לשטח רק רמה אחת.

**.sort**

משנה את המערך וממיינת אותו מיון לקסיקאלי. מיון סטרינגים נעשה נכון כברירת מחדל. כדי למיין מספרים צריך להעביר למתודה פונקציה עם משתנים להשוואה:

arr.sort((a, b) => a - b); // - סדר עולהascending order

arr.sort((a, b) => b - a); // - סדר יורדdescending order

לא יעבוד על מערך המכיל מספרים וסטרינגים יחד.

**בניית/יצירת מערך חדש**

new Array(1, 2, 3) // [1, 2, 3]

new Array(3) // [empty x 3] -

המערך שיווצר לא יהיה ניתן למילוי אלא באמצעות מתודת ה-fill:

arr.fill(1) – ימלא את המערך בערך שצויין.

arr.fill(1, 3) – 1 הוא הערך שיוכנס, 3 הוא האינקס ההתחלתי.

arr.fill(1, 3, 5) – 5 הוא האינדקס האחרון – לא כולל.

**.from**

יוצרת מערך חדש שהוא העתק בעומק אחד (לא יכלול את האובייקטים אליהם יש הפניות מאיברים במערך הנתון) של המערך או אובייקט איטרבילי המועבר כפרמטר.

Array.from({length: 7}, (\_, i) => i + 1); // [1, 2, 3, 4, 5, 6, 7]

Array.from({length: 7}, ( ) => 1); // [1, 1, 1, 1, 1, 1, 1]

Array.from(document. querySelectorAll('.main'), el => Number(el.textContent.replace(x, y));

**הוספת אלמנטים למערך**

**.push**

* מכניס אלמנט לסוף מערך (יעיל וחסכוני מבחינת ביצועים)
* מחזיר את האורך של המערך המעודכן

**.unshift**

* דוחף אלמנט לתחילת המערך (מאוד יקר במשאבי מערכת וזמן ריצה)
* מחזיר את האורך של המערך המעודכן

**הסרת אלמנטים**

**.pop**

* מסיר את האלמנט האחרון – אין צורך להעביר ארגומנט - .pop() (יעיל וחסכוני)
* מחזיר את האלמנט שהוסר

**.shift**

* מסיר את האלמנט הראשון – אין צורך להעביר ארגומנט (לא חסכוני)
* מחזיר את האלמנט שהוסר

**מציאת אלמנט במערך**

**.indexOf(element)**

* מחזירה את האינדקס של האלמנט
* מחזירה -1 אם האלמנט לא נמצא במערך
* **מתודה ישנה שכיום פחות בשימוש**

**.includes(element)**

* מחזירה ערך בוליאני אם האלמנט נמצא או לא במערך
* משתמשת בשוויון קשיח (===)
* **כיום מחליפה את indexOf**

**מתודות סטרינגים**

**str.lastIndexOf**

* מחזיר את האינדקס במופע האחרון של האלמנט

**\* ניתן לעשות indexOf למילה שלמה**

**str.slice(4)**

מחזיר סטרינג חתוך מהנקודה שציינו והלאה (אם לא ציינו אינדקס סופי).

אם ציינו אינדקס סופי נקבל סטרינג חדש מנק' ההתחלה עד לנק' השנייה (לא כולל).

**str.slice(4,8)**

האורך של הסטרינג החדש תמיד יהיה הסוף פחות ההתחלה.

שיטה למציאת התו האחרון בסטרינג –

**str.slice(-1)**

(כאשר מציינים מספר שלילי הספירה נעשית מהסוף)

**Capitalization**

const name = 'mOsHe';

const nameLower = name.**toLowerCase**();

const correctName = nameLower[0].**toUpperCase**() + nameLower.**slice**(1);

**הסרת רווחים וירידות שורה**

str.trim()

כדי להסיר רווחים רק מהסוף או רק מההתחלה:

str.trimStart()

str.trimEnd()

**החלפה**

str.replace('x', 'y')

ניתן גם להחליף מילים שלמות.

**replace מחליפה רק את המופע הראשון של התו או של התווים שניתנו**

כדי להחליף את כל המופעים:

str.replace(**/x/g**, 'y')

**מתודות שמחזירות true או false**

str.includes('x')

str.startsWith('x')

str.endsWith('x')

**str.split מקבלת תו לפיו הסטרינג יומר למערך**

'Ori Baram'.split(' ') // ['Ori', 'Baram']

**.join מקבלת מערך ומחברת לכדי סטרינג כאשר התו שצויין מפריד בין הסטרינגים**

['Ori', 'Baram'].join(' ') // 'Ori Baram'

**הוספה אחידה (padding)**

str.padStart(25, '+') – יוסיף + בהתחלה **עד שהסטרינג יהיה באורך שצויין** (25) – המספר מציין את האורך החדש של הסטרינג.

יישום שימושי – הסתרת המספרים הראשונים בכרטיס אשראי:

const maskCreditCardNamber = number => {

// Convert number to string

const str = number + "";

const last4Nambers = str.slice(-4);

return last4Numbers.padStart(str.length, '\*');

}

**חזרה**

**.repeat**

const chorus = "Because I'm happy";

console.log(`Chorus lyrics for "Happy": ${chorus.repeat(27)}`);

**חיבור סטרינגים**

const str1 = 'Hello';

const str2 = 'world';

const str3 = str1.concat(' ', str2) // 'Hello world'

const str4 = str1.concat(', ', str2) // 'Hello, world'

**מספרים**

דרכים להמרת סטרינג למספר:

Number('32')

+'32'

Number.parseInt('30px') // 30

Number.parseFloat('2.5rem') // 2.5 – כולל מספרים אחרי הנקודה העשרונית

כדי לבדוק אם ערך הוא אינסופי או מספר עדיף **לא להשתמש** ב-isNaN, מכיוון שהוא אינו מדוייק בכמה מקרי קצה ולכן עדיף להשתמש ב-Number.isFinite לבדיקה אם מדובר במספר אין סופי (תוצאה שלילית – מספר סופי, תוצאה חיובית, מספר אין סופי) או עם Number.isInteger לבדיקה אם מדובר במספר.

**Math.sqrt(x)**

שורש ריבועי

Math.max(5, 18, 2, 9, 7) – מספר מקסימלי. לא עובד עם מערכים ולכן נפרוס את המערך כדי לעבוד עם המתודה: Math.max(…arr)

כנ"ל Math.min – לקבלת המספר הקטן ביותר.

**חישוב שטח מעגל:**

Math.PI \* Number.parseFloat('10px') \*\*2;

**חישוב רדיוס:**

Math.PI \* Number.parseFloat('10px');

**מספר רנדומלי**

בין 1 ל-6

Math.floor (Math.random()\*6) + 1;

**פונקציה לחישוב מספר רנדומלי בין 2 מספרים נתונים**

const randomInt = (min, max) => Math.floor(Math.random( ) \* (max – min) + 1) + min;

**עיגול מספרים**

Math.trunc(56.69) – רק מורידה את המספרים אחרי הנקודה בלי חוקיות כלשהי

Math.round(29.8) – מעגלת לאינטג'ר הקרוב

Math.ceil(52.1) – תמיד מעגלת כלפי מעלה

Math.floor(46.9) – תמיד מעגלת כלפי מטה

\* במספרים שליליים העיגול יהיה בהתאם לחוקיות ההופכית במספרים שליליים ועיגול מספר שלילי כלפי מטה יראה ככה למשל – Math.floor(-56.8) => 57-

**הוספת מספרים אחרי הנקודה העשרונית**

(2.7).toFixed(3) // 2.700 (String)

התוצאה שתינתן תהיה סטרינג ולא מספר

כדי להמיר את התוצאה למספר:

+(2.347).toFixed(2) // 2.34 (number)

**פונקציה לבדיקת זוגי ואי זוגי (מודולו)**

const isEven = n => n % 2 === 0;

שימושי כשרוצים לעשות משהו לסירוגין:

If(I % 2 === 0) do something

**BigInt**

אינטג'רים רגילים יכולים להגיע עד לגודל הבא:

2 ^ 53 – 1 = 9007199254740991 === MAX\_SAFE\_INTEGER

אם צריכים לאחסן מספרים מאוד גדולים אז מוסיפים למספר הממש גדול n בסוף:

85679108001235678n

או

BigInt(85679108001235678)

לא ניתן לחשב מספרים רגילים יחד עם BigInt אז ממירים מספרים ל-BigInt:

const crazyBigNumber = 85679108001235678;

const number = 23;

crazyBigNumber + BigInt(number)

באופרטורי השוואה ניתן לשלב מספרים רגילים ו-BigInt

85679108001235678n > 35 // true

לא ניתן לעבוד עם Math ב-BigInt.

BigInt לא יכול להיות עם נקודה עשרונית – הוא תמיד יהיה אינטג'ר.

**מספרים בינלאומיים:**

const options = {

style: 'unit'/'percent'/'currency',…

unit: ' kilometer-per-hour'/'mile-per-hour'/'celsius',…

currency: 'EUR'/'USD'/'ILS'

useGrouping: true, - פסיקים באלפים, כן או לא

}

new Intl.NumberFormat('he-IL', options).format(number);

קודים של מטבעות:

<https://www.iban.com/currency-codes>

**תאריכים ושעות**

4 דרכים ליצירת תאריך חדש:

עכשיו => new Date()

תאריך מסוים => new Date('December 24, 2020')

New Date(2037, 10, 19, 15, 23, 5)

שניות, דקות, שעות, יום בחודש, חודש (מינוס 1 כי זה על בסיס 0), שנה

מילישניות מאז 1970 => new Date(1618212323641)

const now = new Date();

now.getFullYear();

על בסיס 0 – יחזיר חודש מינוס 1 - now.getMonth();

יום בחודש – now.getDate();

יום בשבוע – על בסיס 0 – יום מינוס 1 – now.getDay();

now.getHours();

now.getMinutes();

now.getSeconds();

מחזיר מילישניות מאז 1970 – now.getTime();

בנוסף לכל המתודות הנ"ל, יש מתודות **set** שבאמצעותן ניתן לקבוע שנה (setFullYear), יום בחודש (setMonth), יום בשבוע (setDay) וכו'

יחזיר ייצוג של תאריך בינלאומי סטנדרטי - now.toISOString();

מחזיר מילישניות מ-1970 עד עכשיו – Date.now()

המרה למילישניות:

(24 \* 60 \* 60 \* 1000)

שעות, דקות, שניות, מילישניות

**רשימת קודים בינלאומיים**

<http://www.lingoes.net/en/translator/langcode.htm>

ייצוג תאריך בפורמט מקומי – now.toLocaleString(' he-IL') // 12.4.2021, 10:37:43

**מציאת הקוד הבינלאומי של המשתמש בדפדפן לפי מיקומו הגיאוגרפי:**

const locality = navigator.language;

הצגת תאריך בפורמט מסוים:

var options = {

weekday: 'long',

year: 'numeric',

month: 'long',

day: 'numeric',

}

new Intl.DateTimeFormat('he-IL', options).format(date) // יום חמישי, 20 בדצמבר 2012

**Timers**

~~const timer = setTimeout(( ) => someFunction( ), 3000);~~

const timer = setTimeout(( ) => someFunction, 3000);

\* לא קוראים לפונקציה עם (), רק שמים אותה פה!

ביטול הטיימר:

if(condition) clearTimeout(timer);

משהו שיקרה כל X זמן – setInterval(( ) => someFunction, 1000);

**לולאות**

**continue**

כאשר יש קטע קוד שלא מעוניינים שיתבצע בלולאת for אבל שהלולאה תמשיך לרוץ, או במילים אחרות, כאשר מעוניינים שהלולאה תדלג על איטרציה בהתאם לתנאי מסוים – משתמשים ב-**continue**

for(…){

If(condition) continue;

Some code…;

}

**break**

כשמעוניינים שהלולאה תעצור בתנאי מסויים.

for(…){

If(condition) break;

Some code…;

}

\* לא ניתן לעשות continue או break בלולאת forEach

**לולאת for of עם אינדקס**

for(const [index, element] of arr.entries()){…}

**אובייקטים**

**2 דרכים לגשת לאלמנט באובייקט –**

1. Object.key
2. Object[key] (מקבל סטרינג או ביטוי משתנה)

**הוספה לאובייקט:**

Object.newKey = value

Object[newKey] = value

**הגדרת פונקציה כמתודה של אובייקט**

const obj = {

funcName: function(x){

some code…

}

**גישה למתודה של האובייקט**

obj.funcName(2);

or

obj["funcName"](2);

**this keyword**

בתוך האובייקט **this** מתייחס לאובייקט עצמו.

כך קוראים לאובייקט בתוך האובייקט.

השימוש במילת המפתח **this** שומר על עיקרון **DRY – Don't Repeat Yourself** שמונע חזרתיות ומונע צורך לשכתב קוד במקרה של שינויים.

נניח מקרה שבאובייקט יש פונקציה שמחשבת גיל ובאובייקט לא הגדרנו מפתח של גיל, ניתן להגדיר בפונקציה עצמה את המפתח ולהשתמש בו באמצעות מילת המפתח **this** – דוגמה:

const obj = {

name: 'Mike',

birthYear: 1998,

clacAge: function(currentYear){

this.age = currentYear – this.birthYear;

return this.age;

}

}

**העתקת אובייקט**

const obj1 = {

name: 'Jhon',

age: 26,

friends: ['Karla', 'Merry', 'Sam'],

}

const obj2 = Object.assign({}, obj1)

**חשוב לדעת** שהעתקה כזאת היא **Shallow copy** – מה שאומר שאם יש בתוך האובייקט שאותו מעתיקים אובייקטים, מערכים ועוד מבני נתונים כאלה, הכתובת שלהם בהעתק תמשיך להפנות למבני הנתונים האלה שקיימים באובייקט המקורי – כלומר, לא תתבצע העתקה פנימית של מבני הנתונים האלה, אלא הפנייה לאותה כתובת בזיכרון, כאמור ושינויים מתוך ההעתק ישנו את המקור.

**לולאות באובייקטים**

Object.keys(obj)

מחזיר מערך של סטרינגים עם שמות ה-keys.

Object.values(obj)

מחזיר מערך של הערכים (values) ושומר על type המקורי שלהם.

Object.entries(obj)

מחזיר מערך של מערכים – כל מערך מייצג צמד של key-value באובייקט, כאשר הראשון בצמד הוא סטרינג של ה-key והשני בצמד הוא ה-value ב-type המקורי שלו (מספר, סטרינג וכו').

const obj = {

name: 'Ori',

age: '42',

}

const entries = Object.entries(obj);

for(const [key, value] of entries){

console.log(`My ${key} is ${value}`);

}

// output => **'My name is Ori'**

// **'My age is 42'**

**מבני נתונים נוספים ב-JavaScript**

**Sets**

מכיל רק אלמנטים ייחודיים === שאינם חוזרים על עצמם

const arrayNumbers = [1, 1, 2, 2];

const setNumbers = new Set(arrayNumbers); // {1, 2}

מתודות:

.size

.has(value)

.add(value)

.delete(value)

.clear()

Set משמשת בעיקר כדי לגלות אם אלמנט קיים במערך או לא באמצעות מתודת has, ספירת אלמנטים ייחודיים במערך (new Set(arr).size) וכמו כן משמשת להסרת ערכים כפולים במערך:

[…new Set(arrayNumbers)];

מכיוון ש-set היא מבנה נתונים איטרבילי, ניתן לפרוס אותה באמצעות אופרטור spread וכן ניתן לעבור על ה-set בלולאות for of ו-forEach מאותה הסיבה.

ניתן לספור תווים ייחודיים בסטרינג באמצעות set:

new Set(str).size;

**Map**

מבנה נתונים של key-value, שבשונה מאובייקטים, ה-keys יכולים להיות מכל סוג של type ולא רק סטרינגים כמו באובייקטים.

**שימוש:**

const m = new Map();

m.set(true, 'correct');

מתודת ה-set מחזירה את ה-map כולו ולכן ניתן לשרשר מתודות set זו לזו ולבנות את ה-map בבת אחת.

m.set(true, 'correct').set(false, 'try again')

דרך אחרת לאכלס map היא על ידי העברת מערך של מערכים כשכל מערך מכיל זוג של key ו-value:

const questions = new Map([

['question', 'What is the best programming language?'],

[1, 'Java'],

[2, 'JavaScript'],

['Correct', 2],

[true, 'You are correct!'],

[false, 'You are wrong!']

]);

**מתודות:**

**.get**

מקבלת את ה-key ומחזירה את ה-value (חשוב להקפיד על ה-type של ה-key)

m.get(true) // 'correct'

השימוש ב-key בוליאני ב-map מאפשר העברה של תנאי למתודת ה-.get

**דוגמה:**

**(מבנה של map):**

const restaurant = {

'open' => 11,

'closed' => 23,

true => 'We are open',

false => 'We are closed',

}

const isOpenNow = time => {

return restaurant.get(time > restaurant.get('open') && time < restaurant.get('closed'))

}

.has(x)

.delete(x)

.size

.clear()

.entries() – מחזירה אובייקט איטרבילי שמכיל את זוגות ה-key-values לפי סדר הוספתם.

.keys() – מחזיר אובייקט איטרבילי של ה-keys (לא מערך) שניתן לפרוס למערך ב-spread.

.values() – מחזיר אובייקט איטרבילי של ה-values שניתן לפרוס למערך ב-spread.

הקצאת אובייקטים כ-keys ב-map שימושי ב-DOM manipulation.

ניתן לקבל את האובייקט (או מערך) באמצעות .get רק לאחר שהקצנו את האובייקט (או את המערך) למשתנה ואת המשתנה נעביר ל-.get (מכיוון שהמשתנה מכיל את ההצבעה למקום בזיכרון של האובייקט או של המערך)

Object.entries(obj)

מחזיר מפה!

המרת אובייקט למפה –

const n = new Map(Object.entries(obj));

לולאה על מפה:

for(key, value] of questions){…}

\* ניתן לעבור על המפה גם בלולאת forEach

המרת מפה למערך נעשית באמצעות spread operator

const arr = […questions];

**פונקציות**

**.bind**

const newFunc = func.bind(obj);

מתודה שמחזירה פונקציה חדשה כאשר ה-this מקובע לאובייקט שנמסר למתודה.

.bind יכולה גם לקבע ארגומנטים => כאשר רק רוצים לקבע ארגומנטים במקום האובייקט מעבירים null.

func.bind(null, 1, 2);

**IIFE – Immediately Invoked Function Expression**

בדרך כלל מיושם בפונקציות שצריכות לרוץ רק פעם אחת במהלך הקוד.

(function() {

…do something…

})();