JavaScript

ספר קורס

מהדורה שלישית. תשפ"ד

פרק מספר 1

תכנות בסיסי בשפת JavaScript

טיפוסים

משתנים וקבועים

לולאות

פונקציות

מערכים

פונקציות ES6 על מערכים

אובייקטים ומחלקות

קצת רקע

שפת JavaScript, או בשם החיבה הנחמד JS, היא שפת תכנות בצד לקוח. כלומר, בדומה לשפות HTML + CSS, זו שפה שרצה (בד"כ) בדפדפן ולא על השרת. מה שמאפשר לה לבצע דברים ששפות צד שרת לא מסוגלות לבצע. בעיקר אינטראקציה על הגולש.

הבסיס דומה לתכנות בסיסי בשפות תכנות אחרות כמו C# או JAVA, ולכן בחוברת זו, ובלימודים בכלל, ניתן יותר דגש על המאפיינים הייחודיים של השפה (לדוגמה אירועים), ולא על תכנות בסיסי, שאותו אנחנו מכירות בשלב הזה משפות אחרות שכבר למדנו.

כלומר, נשים דגש יותר על "איך כותבים" מאשר "מה המשמעות". לדוגמה בלולאות for, בנושא של if, פונקציות מתמטיות וכדו'.

הטמעת קובץ JS בעמוד

כאשר אנחנו מריצות עמוד או אתר, אנחנו מריצות תמיד עמוד HTML.

כאשר הדפדפן מקבל את קובץ ה HTML הוא סורק אותו כדי לזהות אילו משאבים נוספים נדרשים להצגה תקינה של העמוד.

משאבים יכולים להיות קבצי CSS, תמונות ומדיה אחרת, פונטים, וכן קבצי JS.

לכן עלינו לזכור להטמיע את קבצי הJS שלנו בקובץ הHTML, בדיוק כמו שהטמענו קבצי CSS. קובץ שלא הוטמע - העמוד לא ישתמש בו.

אופן הטמעת קובץ JS:

```
<script src="scripts.js" type="text/javascript"></script>
```

המאפיין src יכיל את שם הקובץ והנתיב אליו (במידה ונדרש נתיב כמובן).

על פי רוב נטמיע את הקובץ בסוף הbody ולא בhead כמו שמקובל בCSS.

יותר מקובץ JS אחד בעמוד

יש אפשרות להטמיע יותר מקובץ JS אחד בעמוד HTML. במידה ועשית כך - הקבצים יכירו האחד את השני.

כך שתוכלי להצהיר על מערך נתונים בקובץ X, ולהשתמש בו בקובץ Y. כנ"ל לגבי שימוש בפונקציות וכדו'.

אבטחת מידע

מאחר ושפת JS רצה בדפדפן ולא בשרת - כל הגולשים של האתר יכולים לקרוא את הקוד שאנחנו כותבות. כולל ההערות שנשאיר בקוד.

לכן **לעולם** לא נשמור שמות משתמש, סיסמאות api keys או כל נתונים חסויים אחרים בקוד JS!

או במקרה של מערך שהוגדר עם מספר תאים מסויים, אבל לא כל התאים מאוכלסים. תאים שנוצרו אבל עדיין לא אוכלסו - יכילו את הערך undefined.

מערך - Array

מעין רשימה של ערכים. במערך הנתונים מסודרים לפי הסדר שבו נכנסו לתוך המערך. על פי רוב כל הערכים במערך יהיו מאותו סוג אבל בשפת JS זו לא חובה, וערכים יכולים להיות מטיפוסים שונים.

התא הראשון במערך יושב במקום ה-0, בדומה לשפות תכנות אחרות.

ל"מקום" שבו נמצא הערך קוראים בשפה המקצועית "אינדקס".

אובייקט - Object

יפורטו בפרק נפרד.

אובייקט הוא טיפוס שאיננו פרימיטיבי. הוא מסוגל להכיל יותר מנתון אחד, אפשר להפעיל עליו פונקציות שונות.

בשפת JS מערכים הם גם כן סוג של אובייקטים, ולא טיפוס משל עצמם.

פונקציה - function

בשפת JS פונקציות הינן טיפוס של נתונים, בדיוק כמו מחרוזת, מספר או אובייקט. יש לזה השלכות בצורת הכתיבה של פונקציות ושימושים שונים של פונקציות, שקיימות בשפת JS ולא קיימות בשפות אחרות. עוד על כך מפורט בפרק על פונקציות.

בדיקת סוג המשתנה

כדי לבדוק מהו הטיפוס של ערך מסויים - נשתמש באופרטור typeof.

לדוגמה:

```
console.log(typeof 42); // "number"
console.log(typeof 'blubber'); // "string"
console.log(typeof true); //"boolean"
console.log(typeof undeclaredVariable); //"undefined"
```

כמובן נוכל גם להריץ את הפקודה גם על משתנים. לדוגמה אם קיים משתנה בשם myVar, נוכל לכתוב:

```
console.log(typeof myVar);
```

ובתגובה נקבל את הטיפוס של הערך שבתוך המשתנה.

פונקציות המרה

קיימות מגוון של פונקציות שמסוגלות להמיר ערכים לטיפוסים אחרים. נתעכב כאן על 2 הפונקציות השימושיות ביותר.

המרה למחרוזת

הפונקציה toString מסוגלת להמיר ערכים של רוב הטיפוסים - למחרוזת. כולל מערכים.

לדוגמה:

```
number = 1234;
console.log(number.toString()); // "1234"
arr = ["apple", "banana"];
console.log(arr.toString()); // "apple,banana"
```

שימי ❤️ שכשזה מגיע לאובייקטים - הפונקציה לא יודעת מה לעשות, ותחזיר ערך לא תקין:

```
let obj = { key: "value" };
console.log(obj.toString()); // "[object Object]"
```

המרה למספר

בהרבה מצבים אנחנו מקבלות ערכים מהגולש, כולל ערכים מספריים. בחלק מהמצבים, כפי שנלמד בהמשך, הערך שמתקבל מהגולש מגיע אלינו כסטרינג למרות שמדובר בספרות בלבד.

במקרה כזה, נרצה להמיר את הספרות למספר, על מנת שנוכל לערוך על המספר חישובים שונים.

לרשותנו עומדות כמה דרכים להמיר ערך כלשהו למספר:

- **parseInt()** - המרה למספר שלם. הפונקציה מסוגלת להמיר גם מספר עם נקודה עשרונית למספר שלם.

```
let number = prompt("Give me a number!");
console.log(number); // "2"
console.log(number + number); // "22" - משורשר כסטרינג
number = parseInt(number);
console.log(number + number); // 4 - פעולת חיבור על מספר

const price = 5.9;
console.log(parseInt(price)); // 5
```

- **parseFloat()** - המרה למספר עשרוני. לדוגמה:

```
let price = prompt("What is the price?");
```

תחביר ופקודות בסיסיות

בפרק זה נכיר כמה מהפקודות הבסיסיות של JS, שישמשו אותנו כדי להדפיס נתונים, לקבל קלט מהגולש (לפחות בשלבים הראשונים) וכן להציג הודעות וערכים חשובים.

תבנית טקסט - text template

שרשור סטנדרטי בJS עובד בדומה לשרשור בשפות תכנות אחרות. לדוגמה:

```
const language = "JavaScript";
const str = "Yay! we started learning " + language;
```

כתוצאה נקבל את המשפט המורכב מהטקסט הקבוע בתוספת המשתנה.

באופן הזה נוכל לשרשר כמה סטרינגים, להוסיף לשרשור חישובים שונים ועוד.

אבל לצורת העבודה הזו יש כמה חסרונות, להלן 2 הבולטות במיוחד:

- לעיתים קרובות יש צורך לשרשר יותר ממשתנה/חישוב אחד. שלא לדבר על מצב שבו רוצים להכניס גם מרכאות כחלק מהשרשור. מצבים אלו ואחרים יוצרים קוד מבולגן ולא קריא.
- בשרשור רגיל אין אפשרות לעבור שורה. כאשר מדובר בסטרינג ארוך מאד (וניצור בהמשך בעז"ה סטרינגים ארוכים למדי) - זה הופך להיות די מסורבל. כדי לעבור שורה צריך לסגור את הסטרינג, להוסיף פלוס (+), לעבור שורה ולהמשיך את הסטרינג שם.

תבנית טקסט פותרת לנו את 2 הבעיות הללו. היא מאפשרת יצירת סטרינג הכולל חלקים קבועים בשיבוץ משתנים, קבועים או חישובים לתוכו בצורה נוחה ביותר, ומאפשר גם מעבר שורה בקוד בלי שום בעיה.

לדוגמה:

```
const adjective = "amazing";
const str = `JavaScript is ${adjective}!`;
```

התו שפותח וסוגר את הסטרינג הוא גרש הפוך. אפשר למצוא אותו במקלדת בצד שמאל למעלה, ממש מתחת לכפתור הEsc, ליד המספר 1.

כדי לשרשר משתנים, קבועים או חישובים פנימה - נכתוב דולר צומדיים, ובתוך הצומדיים נכניס את המשתנה, הקבוע או החישוב.

להלן דוגמה נוספת:

```
const cookiesAte = 7;
const cookiesBaked = 15;
console.log(`I baked ${cookiesBaked} cookies, but I accidentally
ate ${cookiesAte}.
```


פונקציות

פונקציות אלו קטעי קוד שסגורים כבלוק. פונקציות קיימות בכל שפות התכנות בצורה כזו או אחרת. בשפת JS ההסתכלות על פונקציות קצת שונה מאשר בחלק מהשפות האחרות שיצא לך להכיר, ועוד על זה - מיד בהמשך.

נשתמש בפונקציות בכמה מקרים. ביניהם:

- **קטעי קוד שחוזרים על עצמם** - במקום לכתוב את הקוד שוב ושוב, נכתוב פעם אחת פונקציה שמבצעת את הפרוצדורה הדרושה וכל מקום שהפרוצדורה הזו נדרשת - רק נקרא לפונקציה.
- **קטעי קוד שמבצעים פעולה ספציפית** - הרבה פעמים נוח לחלק קטן קוד ארוך לפונקציות קצרות, שכל אחת מבצעת פעולה ספציפית. לדוגמה פונקציה שמגרילה תא ממערך, פונקציה שמקבלת מערך ויודעת לצייר אותו, וכו'. באופן הזה אנחנו שומרות על קוד מסודר וברור.
- **ארגומנטים של פונקציות אחרות** - כמו שנראה מיד, בשפת JS יש אפשרות לשלוח פונקציות כארגומנטים של פונקציות אחרות. לדוגמה הפונקציה sort שממיינת מערך. יש לה את התנהגות ברירת המחדל - מיון אלפביתי, אבל אם רוצים מיון אחר - אפשר לשלוח בסוגריים של הפונקציה sort פונקציה נוספת שנכתוב, והיא מגדירה לsort לפי מה למיין את המערך.

פונקציה היא סוג של נתונים

בניגוד לשפות אחרות, JS מתייחסת לפונקציה כטיפוס בפני עצמו. בדיוק כמו string, number, array יש לנו טיפוס נתונים בשם function. אפשר לראות את זה כאשר נבקש מJS להדפיס לנו את הטיפוס של הפונקציה באמצעות typeof:

```
function myFunc(){
  return 12;
}
// הדפס את הטיפוס שמחזירה הפונקציה
console.log( typeof myFunc() ); // number
// myFunc הדפס את הטיפוס של המשתנה
console.log( typeof myFunc ); // function
```

כתוצאה מזה JS מאפשרת לנו להכניס פונקציה לתוך משתנה/קבוע, לשלוח פונקציות כארגומנטים של פונקציות אחרות, להגדיר פונקציות כמאפיינים של אובייקטים, ועוד.

חשוב לזכור את הנקודה הזו, כי היא תעזור לנו להבין את המשמעות וצורת העבודה בעז"ה בהמשך.

תחביר סטנדרטי

נעבור בקצרה על הכללים לכתיבת פונקציות וקריאה להן.

```
function compliment() {
  alert("You are doing great!!");
}
```

הצהרנו על פונקציה בשם compliment שמה שהיא עושה זה להדפיס את הטקסט "You are doing great!!!". כל עוד לא קראנו לה - הטקסט לא יודפס.

כדי לקרוא לפונקציה - פשוט נקרא לה בשם:

```
compliment();
```

באותו רגע הפונקציה תתחיל לרוץ וההודעה מחממת הלב תודפס על המסך.

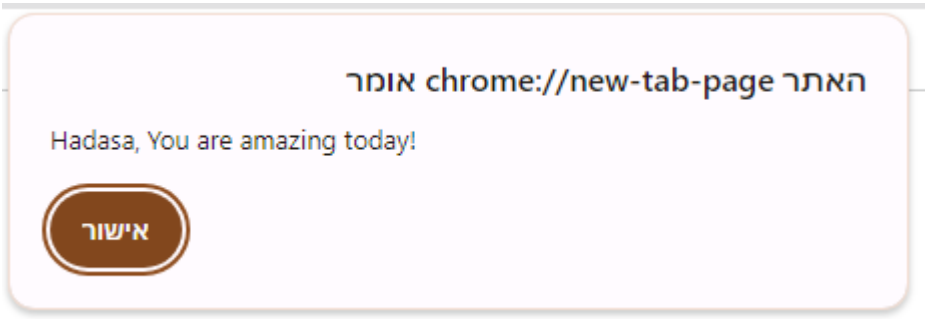
פונקציה שמקבלת ארגומנטים

פונקציות יכולות לקבל נתונים שונים, ואז לבצע את הפעולה שלהן בהתאם לנתונים שהתקבלו.

נתונים אלו מכונים בשפה המקצועים ארגומנטים. לדוגמה:

```
function complimentMe(name){
   alert(`${name}, You are amazing today!`);
}
complimentMe("Hadasa");
```

הפעם נדרשתי לשלוח לפונקציה נתון - השם שלי. הנתון שאותו שלחתי לפונקציה נכנס לתוך משתנה בשם name, ואז הפונקציה יכלה להדפיס לי הודעה מותאמת אישית:



שימי ❤️ שאפשר לשלוח טקסט ישירות, או לשלוח טקסט ששמור בתוך משתנה. הפונקציה complimentMe תכניס את הערך שהתקבל לתוך המשתנה name בלי קשר לאיך שקראו למשתנה במקור.

ניתן כמובן לייצר פונקציות שמקבלות יותר מארגומנט אחד. לדוגמה:

```
function mazalTov(name, gender){
     const text = gender === "boy" ? "His" : "Her";
     alert(`It's a ${gender}!! ${text} name is ${name} 💐`);
```

פונקציות חץ - arrow functions

עד עכשיו עבדנו עם התחביר המסורתי של פונקציות, אבל יש דרך קצרה יותר לכתוב פונקציות בJS.

לדוגמה הפונקציה הבאה:

```
function doubleNum(num){
     return num * 2
}
```

אפשר לכתוב אותה בצורה שונה:

```
const doubleNum = num => num * 2;
```

יצירת פונקצית חץ צעד אחרי צעד

שלב ראשון - הכנסת הפונקציה לתוך קבוע

פונקציות חץ נמצאות בתוך קבוע, ובשלב הראשון נכניס את הפונקציה שלנו לתוך קבוע, בדיוק כמו שלמדנו בפרק הקודם:

```
const doubleNum = function(num) {
  return num * 2;
}
```

שלב שני - מעבר לפונקצית חץ

נוריד את המילה function, ובמקומה נשים חץ (שמורכב מהסימן שווה ואחריו הסימן >). החץ ממוקם בין הסוגריים שמקבלים משתנים לבין הצומדיים שפותחים את "מה הפונקציה עושה":

```
const doubleNum = (num) => {
  return num * 2;
}
```

בשפת JS אנחנו מזהים פונקציות לפי המילה function או לחילופין, לפי הסימן של החץ. כאשר את נתקלת בחץ בקוד - סימן שיש כאן פונקציה.

שלב שלישי - אופציונלי - הורדת הסוגריים

כאשר פונקצית חץ מקבלת ארגומנט אחד בלבד - לא 0 ארגומנטים ולא 2 ארגומנטים - אפשר להוריד את הסוגריים שמסביב לשם הארגומנט:

```
const doubleNum = num => {
```


indexOf/lastIndexOf - איתור ערך בתוך מערך

הפונקציות עובדות בצורה דומה לindexOf/lastIndexOf על מחרוזות, כך שהן מקבלות בסוגריים ערך, ומחזירות את האינדקס של התא הראשון שבו הערך מופיע.

ההבדל היחיד ביניהן הוא ש-indexOf מתחילה מהתא הראשון, ו-lastIndexOf מתחילה מהתא האחרון.

לדוגמה:

```
let fruits = ['Apple', 'Orange', 'Apple']
alert( fruits.indexOf('Apple') ); // 0 (first Apple)
alert( fruits.lastIndexOf('Apple') ); // 2 (last Apple)
```

הפונקציה תחזיר -1 במידה והערך שחיפשנו לא קיים במערך כלל.

```
alert( fruits.indexOf('Banana') ); // -1
```

שימי ❤️שהפונקציות הללו מחפשות ערכים פשוטים כמו סטרינג או מספר בתוך המערך. ויותר מזה - הפונקציות תחזרנה אינדקס של תאים שמכילים את הערך המבוקש **בשלמותו**, ולא רק חלק ממנו.

במידה ואת נדרשת לאתר על בסיס נתון מורכב יותר - נשתמש בפונקציה find. לדוגמה כאשר אני צריכה לאתר את התלמידה שמספר הזהות שלה הוא 123456789 בתוך מערך שבו כל תא הוא אובייקט של תלמידה.

includes - האם ערך קיים בתוך מערך

הפונקציה עובדת באופן זהה לפונקציות indexOf/lastIndexOf, אלא שהיא מחזירה true/false האם הערך קיים או לא קיים במערך, ולא את האינדקס של התא שאותר:

```
let fruits = ['Apple', 'Orange', 'Peach'];
alert( fruits.includes('Banana') ); // false
alert( fruits.includes('Peach') ); // true
```

שינוי סדר התאים במערך

toSorted/sort - מיון מערך

הפונקציות sort/toSorted מאפשרות מיון של מערכים.

ההבדל היחיד ביניהן הוא ש**sort** ממיינת את המערך עצמו, ולעומתה **toSorted** מחזירה מערך ממויין בלי לשנות את המערך עצמו.

בכל הדוגמאות שלהלן מודגמת sort, אבל אפשר להשתמש באותה מידה גם בtoSorted.

פונקציות מתקדמות על מערכים

מבוא קצר

בקטע זה נלמד על כמה פונקציות של JS שעובדות על מערכים.

את כל מה שעושות הפונקציות הללו אפשר לעשות עם לולאות רגילות, אבל העבודה עם הפונקציות הייעודיות הללו קצרה, קלה וברורה יותר. לכל פונקציה יש את המטרה שלה, את הcallback שלה ואת מה שהיא מחזירה, אבל כולן כתובות באותו תחביר.

הפונקציות הללו נוספו לשפת JS החל מגרסה ES6 וחלק מהן אפילו טיפה מאוחר יותר.

רק על מערכים!

שימי ❤️ שהפונקציות יעבדו אך ורק על מערכים. לא על אובייקטים, לא על רשימת אלמנטים ששאבנו מהHTML באמצעות querySelectorAll וכו'.

אם יש אלמנט רשימה שאינו מערך, ואנחנו רוצות להפעיל עליו אחת מהפונקציות הללו - עלינו להמיר אותו למערך. לדוגמה:

```
const buttons = document.querySelectorAll("button");
```

הפונקציה querySelectorAll מחזירה אובייקט מסוג NodeList - רשימת "ענפים", ולא מערך רגיל. כדי להמיר את הרשימה למערך נוכל להשתמש בפונקציה Array.from() או באופרטור spread. להלן דוגמה לשימוש בכל אחת מהאפשרויות הללו:

```
let buttonsArray = Array.from(buttons);
let buttonsArray = [...buttons];
```

בכל אחד מהמקרים המשתנה buttonsArray יכיל מערך רגיל, עם הנתונים שקודם לכן היו בתוך הרשימה מסוג NodeList.

טיפוסים נוספים שידרשו את ההתאמה הזו הם לדוגמה אובייקט מסוג Set, שהוא אובייקט שדומה למערך, אבל לא מסוגל להכיל ערכים כפולים, או סטרינג. במידה ונרצה להפעיל את אחת מהפונקציות בפרק זה על סטרינג - נצטרך להמיר אותו קודם לכן למערך.

לולאת forEach

הקדמה - זו לולאה שלא רצה לפי תנאי, אלא רצה על מערך/אובייקט, בכל איטרציה (פעימה) היא עובדת על מקום אחד במערך.

לדוגמה:

```
let myArray = ['first', 'second', 'third'];
```



```
myArray.forEach(my_func);

function my_func(value, index) {
    console.log(`value: ${value}, index: ${index}`);
}
```

תחביר:

אנחנו קוראות לפונקצית לולאה שפועלת על המערך, ומקבלת כארגומנט פונקציה **(callback function)**. הלולאה תרוץ על כל תאי המערך לפי הסדר באופן אוטומטי.

הפונקציה הפנימית - callback - מקבלת כארגומנטים את ה**ערך** ואת ה**אינדקס** - המקום במערך. אין חובה לקבל אף אחד מהם.

אפשר להשתמש בפונקציות אנונימיות ואפשר בפונקציות רגילות.

```
let myArray = ['first', 'second', 'third'];
myArray.forEach(function(value, index) {
  console.log(`value: ${value}, index: ${index}`);
});
```

ואפשר גם להשתמש כאן בפונקציות חץ.

```
let myArray = ['first', 'second', 'third'];
myArray.forEach((value, index) => {
  console.log(`value: ${value}, index: ${index}`);
});
```

שימי ❤️ שבשורה האחרונה אנחנו סוגרות קודם את הבלוק של הפונקציה האנונימית ואחר כך את הסוגריים של הפונקציה forEach.

map

הפונקציה map רצה על מערך ומחזירה מערך חדש המבוסס על המערך המקורי. הפונקציה מקבלת בתוך הסוגריים פונקציה נוספת (callback function), שמגדירה מה הערך שיוחזר עבור כל תא במערך המקורי.

לדוגמה:

```
const prices = [2, 60, 4];
const new = prices.map(function (value) {
    let newValue = value * 4;
    return newValue;
```

אובייקטים

אובייקטים הם משתנים שמכילים הרבה מאד מידע סביב נושא ספציפי.

לדוגמה כל המידע על משתמש, מוצר, קלף במשחק וכדו'.

כל נתון באובייקט מורכב ממפתח (key) וערך (value). ערכים יכולים להיות מכל טיפוס שהוא, כולל אובייקטים או מערכים ואפילו פונקציות.

הבה ניצור אובייקט מסוג משתמש:

```
const myUser = {
  id: 123456789,
  name: 'Noa'
}
console.log(myUser);
```

וזה מה שנקבל בקונסול:

```
▶ {id: 123456789, name: 'Noa'}
```

לאובייקט שיצרנו יש שני מאפיינים - מספר זהות ושם. כאשר מדפיסים את המשתנה לקונסול - נוכל לראות שהטיפוס הוא Object, ואת המאפיינים שלו.

קריאת/הוספת/עדכון ערכים באובייקט

כדי לערוך או לא להוסיף ערך בתוך אובייקט עלינו להשתמש במפתח. לדוגמה:

```
user.name = 'Noa Cohen';
user.friends = ["Shira", "Riki"];
user.parents = { father: "Moshe", mother: "Rachel"};
```

כך גם אפשר להדפיס ערכים או לקרוא לפונקציות שהם ערכים באובייקט, וכן לרוץ בלולאות על ערכים שהם מערכים, בדיוק כמו שעובדים עם מערך רגיל. לדוגמה:

```
console.log(user.name);
for(let friend of user.friends){
  console.log(`Hi ${friend}`);
}
```


שם מפתח בתוך משתנה

כאשר שם המפתח שמור לנו בתוך משתנה - נוכל לגשת למפתח עם סוגריים מרובעים ולא עם נקודה, בדיוק כמו שניגשים לערך בתוך מערך:

```
let key = id;
console.log(user[key]); // 123456789
```

נדגים יצירת פונקציה שמקבלת אובייקט + שם של מפתח, ומחזירה את הערך שנתון באובייקט תחת אותו מפתח:

```
const getValue = (obj, keyName) => obj[keyName];

const workers = {
  Daniel: "Manager",
  Moshe: "Developer",
  Yaakov: "Designer",
  Avi: "Tester"
};
console.log(`Avi is the ${getValue(workers, "Avi")}`);
// Avi is the Tester
```

מחיקת ערך מאובייקט

כאשר רוצים למחוק מתוך אובייקט איזשהו ערך, כולל המפתח שלו נשתמש בפקודה delete:

```
delete user.friends;
```

שימי ❤️ שבעבודה השוטפת זה פחות שימושי, מאחר ויכול לגרום למקרים בעייתיים של הסתמכות על תכנים שלא קיימים יותר באובייקט ועוד. אבל האפשרות קיימת.

מתודות - פונקציות של אובייקטים

JS מאפשרת לנו לתת לאובייקט פונקציות משלו. פונקציות אלו מכונות בשפה המקצועית **methods - מתודות**.

לדוגמה:

```
const myUser = {
  id: 123456789,
  name: 'Noa',
  sayHello: function(){
```



```
    subject: "JavaScript",
    duration: "3 hours",
    numStudents: 30,
};
const lesson2 = lesson;
lesson2.subject = "C#";
console.log(lesson.subject); // "C#"
```

אז איך בכל זאת לשכפל מערך/אובייקט?

ישנן מספר שיטות שמאפשרות לנו לשכפל מערך או אובייקט כמו שמעתיקים משתנים מטיפוסים אחרים. הנוחה ביותר היום היא **spread**.

spread

הפרוצדורה spread מאפשרת לנו לפצל ערכים של מערך או אובייקט (או סטרינג) לערכים בודדים.

הבה נכיר סיטואציות שבהן אפשר להשתמש בspread, והנושא יהיה ברור יותר באמצעות הדוגמאות.

אלו לא כל השימושים של spread ולא כל האפשרויות שלו, אלא הבסיסיות ביותר.

העתקת מערך

כמו שכתבנו למעלה, כאשר אומרים "x = y" במערכים - המערך החדש רק מצביע על המערך הראשון.

spread עוזרת לנו במה שמכונה "העתקה עמוקה". היא מפצלת את המערך הראשון לערכים בודדים, ואז נכניס את אותם ערכים ישירות למערך החדש. באופן הזה לא העתקנו מערך שלם אלא ערכים בודדים, ועם זה אין שום בעיה, וההעתקה תהיה רגילה, בדיוק כאילו העתקתי ערכים פרימיטיביים (כמו boolean או מחרוזת).

```
const mothers = ["Sara", "Rivka", "Rachel", "Lea"];
const names = [...mothers];
names.push("Yael");
console.log(names); // ["Sara", "Rivka", "Rachel", "Lea",
"Yael'];
console.log(mothers); // ["Sara", "Rivka", "Rachel", "Lea"]
```

בתחילה יצרנו מערך עם שמות. בשלב השני יצרנו קבוע חדש, ואמרנו לו שהוא שווה למערך. מה תהיה תכולת המערך? לך תפרק את המערך mothers לאיברים בודדים ותכניס אותם לתוך המערך החדש.

באופן הזה המערך החדש מנותק מהמערך המקורי, וכאשר הוספתי לו שם חדש - המערך המקורי לא השתנה.

פרק מספר 2

DOM

Document Object Module

עבודה של JS מול הHTML

יצירת אלמנטים חדשים

אירועים

Document Object Module

document הוא אובייקט JS שמחזיק בתוכו את הHTML ואת כל המידע עליו.

כאשר נרצה לגשת לאלמנטים בתוך הHTML - נתחיל מהdocument.

כל הסנכרון בין JS לבין HTML מכונה בשפה המקצועית Document Object Module, או בקיצור DOM.

ישנן 2 פונקציות שמשמשות אותנו כדי לקבל אלמנטים מתוך הHTML:

- **document.getElementById** - מקבלת בסוגריים ID (בלי סולמית).
- **document.querySelector** - מקבלת בסוגריים סלקטור כלשהו של HTML. בפונקציה זו נכתוב את הסלקטור בדיוק כמו שאנחנו כותבות סלקטורים של CSS. אם זה קלאס - ניתן נקודה, אם זה ID - ניתן סולמית, וכו'.

לדוגמה:

```
let my_img = document.getElementById("person_img");
let my_img = document.querySelector("#person_img");
```

שתי השורות הללו מבצעות את אותה פעולה. שימי ❤️ להבדל בסלקטור שבתוך הסוגריים. בשורה הראשונה קוראים לו בלי סולמית ובפעם השניה - עם סולמית, כי השתמשנו בפונקציה querySelector.

שתי הפונקציות הללו מחזירות אלמנט HTML אחד בלבד. גם אם יש יותר מאלמנט אחד שעונה על ההגדרה שבסוגריים.

על פי רוב נעדיף את העבודה עם **querySelector**, מאחר והיא נוחה יותר ומאפשר לנו תמרון גדול יותר בגישה לאלמנטים.

מניפולציות על אלמנטים

כאשר יש לי ביד אלמנט - אני יכולה להתחיל להפעיל עליו מניפולציות שונות.

אפשר לשנות לו את התוכן ואפילו להכניס לתוכו HTML חדש שלם, הכולל תגיות. אפשר לקבל, להוסיף ולשנות את הקלאסים שלו, את העיצוב שלו ומאפיינים אחרים שלו. וכן אפשר להפעיל עליו אירועים.

שימי ❤️ שבמידה וניסית להפעיל מניפולציה על אלמנט שלא קיים בDOM - תחזור לך שגיאה שתופיע גם בקונסול.

קלאסים - classList

כדי לגשת לקלאס/ים של האלמנט עלינו לגשת בשלב הראשון למאפיין classList. המאפיין הזה מחזיר לנו מערך של כל הקלאסים של אותו אלמנט.

לדוגמה


```
    <h3 class="student_name">${student.fullname}</h3>
    <div class="student_grade">${student.grade}</div>
    <img src="${student.profileImg}" />
  </div>`;
});
document.querySelector("#allStudents").innerHTML = output;
```

כמה נקודות לתשומת ❤️:

- מומלץ להשתמש בtext tepmlate כדי ליצור את הסטרינג. הוא מאפשר לנו הן מעבר שורות לצורך קוד קריא ונעים בעין, והן שימוש במרכאות מכל סוג, כמקובל בHTML.
- כאשר כותבים קוד בJS "טהור" - כלומר, בלי ספריות נוספות - לא מקובל להשתמש הרבה בinnerHTML בגלל בעיות אבטחה שהוא עשוי לגרום. במקום זה נשתמש בcreateElement שאותו נלמד בפרק הבא.

כן חשוב להכיר את צורת העבודה הזו, כי כאשר תגיעי בעז"ה ללימודי ריאקט - צורת העבודה דומה יותר לinnerHTML מאשר לכל האפשרויות האחרות שמקובלות בJS רגיל.

onclick - הצמדת אירוע

נושא אירועים בJS נכתב בצורה נרחבת בהמשך ספר זה.

נביא כאן דוגמה פשוטה להצמדת אירוע קליק, לחיצת כפתור, על אלמנט מהHTML.

```
const btn = document.querySelector("#myButton");
btn.onclick = function(){
  alert("hello");
};
```

אפשר להשתמש בפונקציה אנונימית (כמו בדוגמה שלמעלה) או בפונקציה עם שם. וכן אפשר כמובן להשתמש בפונקצית חץ.

דוגמה לפונקציה עם שם:

```
function doSomething(){
  alert("Something happened";
}
document.querySelector("#do").onclick = doSomething;
```

אנחנו כותבות את שם הפונקציה **בלי** סוגריים בסופה. במידה ונכתוב סוגריים - הפונקציה תקרה בטעינת העמוד ולא בעת לחיצה על הכפתור.

אירועים

מאזינים לאירועים

בJS יש 2 דרכים שונים להצמדת אירוע לאלמנטים בHTML.

הדרך הראשונה היא המוכרת לנו - באמצעות on. פשוט קוראים לאלמנט, מוסיפים נקודה, ואז כותבים on וסוג האירוע הדרוש. לדוגמה onclick, onchange.

```
document.querySelector("#element").onclick = function(){
   // Do something here...
}
```

הדרך השניה היא הוספת מאזינים לאירועים. התחביר עובד בצורה הבאה:

```
document.querySelector("#element").addEventListener("click",
function(){
   // Do something here...
});
```

כמו בon, אפשר גם כאן להשתמש בפונקציות אנונימיות (כמו בדוגמה שלמעלה) או בפונקציות עם שם:

```
function do_something(){
   alert('hello');
}
document.querySelector("#btn").addEventListener("click",
do_something);
```

שימי ❤️ שאין צורך לכתוב on לפני סוג האירוע. נכתוב click או change בלבד.

הצמדת יותר ממאזין אחד לאירוע

היתרון העיקרי של האזנה לאירוע בצורה הזו היא שאפשר להגדיר יותר מפונקציה אחת שתקרה בזמן מסויים. לדוגמה - בעת לחיצת כפתור תעשה את הפונקציה X, וגם את הפונקציה Y וכו'.

זה בניגוד לשימוש בon שלמדנו עד היום, שבו ברגע שהגדרתי פונקציה לאירוע - יהיה לאלמנט רק את הפונקציה הזו. לדוגמה:

```
const element = document.querySelector("#btn");
element.onclick = func_1;
element.onclick = func_2;
```

בעת לחיצה על הכפתור - רק הפונקציה func_2 תקרה.

האובייקט event

האובייקט event הוא אובייקט שנוצר באופן אוטומטי כאשר קורה כל אירוע שהוא בעמוד. כדי להשתמש בו יש להצהיר עליו כארגומנט בפונקציה שקורית בעת האירוע. לדוגמה:

```
myBtn.addEventListener('click', function(e){
     console.log(e);
});
```

האובייקט מכיל המון מידע הקשור באירוע, וגם כמה פונקציות שימושיות.

להלן כמה מהשימושים הנפוצים ביותר לאובייקט event:

ביטול אירוע ברירת המחדל - e.preventDefault()

הפונקציה preventDefault מבטלת את מה שהיה צריך לקרות כברירת מחדל.

בעיקר משמשת אותנו כאשר רוצים לבטל שליחה של טופס בצורה הרגילה של HTML (ריענון העמוד או מעבר לקישור שכתוב בaction של הטופס), ורוצים לערב שליחה באמצעות JS, ואלידציות נוספות, AJAX ועוד.

לדוגמה:

```
my_form.addEventListener("submit", function(e){
     e.preventDefault();
     alert("Submit is here!!");
});
```

ביטול ביעבוע של האירוע - e.stopPropagation()

בכל פעם שקורה אירוע כלשהו על אלמנט בעמוד - לדוגמה קליק על כפתור - האירוע מבעבע כלפי מעלה בעץ של הDOM, כך שגם האבא של הכפתור מקבל אירוע קליק, וכן האבא שלו וכן הלאה, עד לBODY.

בדרך כלל זה לא מפריע לנו, אבל יש מקרים שבהם נרצה לעצור את העלאת האירוע להורה של האלמנט, ואז נשתמש בstopPropagation.

לדוגמה - כאשר יוצרים דרופדאון דרך JS, ובנוסף מגדירים שבעת לחיצה על הbody - הדרופדאון ייסגר, כדי לסגור אותו בעת לחיצה מחוצה לו. נצטרך להגדיר stopPropagation על הדרופדאון, כדי שבעת לחיצה **עליו** או **בתוכו** - הקליק לא יבעבע לbody ולא יגרום לו להסגר.

```
document.querySelector("body").onclick(() =>{
  const drop = document.querySelector("#dropdown.open");
  if(drop)
```

פרק מספר 3

נושאים נוספים בJS

שמירות נתוני גולשים

JSON

טיימרים

ביטויים רגולריים - RegEx

אובייקטים נוספים של JS

JSON

מבוא

JSON הינו פורמט שימושי לשמירת והעברת נתונים.

פורמט JSON הוא בעצם סטרינג, שאפשר להמיר באמצעות פונקציה אחת לסטרינג שכולל נתונים, גם מערכים ואובייקטים.

הפורמט הזה מאפשר להעביר נתונים בין צד שרת וצד לקוח, בין שני שרתים ועוד, בצורה נוחה ופשוטה לשני הצדדים.

בJSON אובייקטים מזוהים באמצעות סוגריים מסולסלים, ומערך מזוהה עם סוגריים מרובעים.

גם המפתח וגם הערך יהיו בתוך **מרכאות** (ולא כמו באובייקט רגיל, ששם המפתח לא נמצא בתוך מרכאות). רק מספרים לא יהיה בתוך מרכאות.

שימי ❤️:

חובה להשתמש במרכאות דווקא ולא בסימן גרש. לכן כשאנחנו כותבים JSON, נעטוף את הסטרינג בגרש או בסמן `.

מבנה לדוגמה

```
‘{
"employees":[
{"firstName":"John", "lastName":"Doe"},
{"firstName":"Anna", "lastName":"Smith"},
{"firstName":"Peter", "lastName":"Jones"}
]
}’;
```

בדוגמה הזו קיים מערך בשם employees, המכיל אובייקטים. בכל אובייקט יש מפתח של שם פרטי ומפתח של שם משפחה.

בדרך כלל הJSON יהיה ללא רווחים, ולא במבנה מסודר כמו כאן, אבל זה לא משנה. JS יודע להמיר אותו למשתנים רגילים (מערך/אובייקט) בכל מקרה.

שימי ❤️ שכשיוצרים סטרינג בJS - אסור לעבור שורה סתם ככה. ראי בדוגמה השניה את אופן הכתיבה המדוייק בJS.

- **JSON.parse()** - המרת סטרינג JSON למשתנים רגילים (מערך/אובייקט)

נתחיל בדוגמה

אפשר לדמות את האחסון למקפיא מגירות גדול, ותהליך הכנת עוגת שכבות.

תחילה אני מכינה בתבנית את השכבה הראשונה, ואז צריכה להכניס אותה למקפיא. אז אני פונה לבת החמודה שלי, ואומרת לה "מותק, תפתחי את המקפיא, ותכניסי את התבנית הזו לתוך מגירה ריקה. תדביקי על המגירה מדבקה שכתוב עליה "עוגת שכבות".

עברה חצי שעה, עכשיו צריך להוסיף את השכבה השניה לעוגה. אני שוב פונה לילדונת ואומרת לה "תגשי למקפיא, ותביאי לי משם את מה שבתוך המגירה שכתוב עליה "עוגת שכבות".

ולבסוף, אחרי חצי שעה של שוקולד וקצפת, אני שולחת אותה בשלישית למקפיא. הפעם עם ההוראה "תגשי למקפיא, ולתוך המגירה שכתוב עליה "עוגת שכבות" תכניסי את התבנית עם העוגה המשודרגת".

התהליך בקצרה

1. מכניסים משהו למקפיא, לתוך מגירה ריקה, וכותבים על המגירה תיאור של התכולה.
2. ניגשים למקפיא ושולפים ממנו את התכולה שבתוך המגירה עם המדבקה שרלוונטית לנו כרגע.
3. משדרגים את התכולה המקורית (או מנשנשים ממנה), ומחזירים למקפיא למגירה שעליה המדבקה.

ועכשיו נאמר את זה בשפת המחשב

- המקפיא = localStorage
- הכנסת תכולה = setItem
- הוצאת תכולה = getItem

כלומר התהליך הוא:

1. מכניסים (setItem) משהו למקפיא (localStorage), לתוך מגירה ריקה, וכותבים על המגירה תיאור של התכולה.
   הפונקציה **setItem** מקבלת בסוגריים 2 פרמטרים: מה לכתוב על המדבקה ואחר כך איזה נתון להכניס פנימה.
   ולכן הקוד ייראה ככה:

```
localStorage.setItem("cake", myCake);
```

2. ניגשים למקפיא (localStorage) ושולפים ממנו (getItem) את התכולה שבתוך המגירה עם המדבקה שרלוונטית לנו כרגע.
   הפונקציה **getItem** מקבלת בסוגריים רק את הטקסט שעל המדבקה.
   ולכן הקוד ייראה ככה:

```
let cakeFromFreezer = localStorage.getItem("cake");
```

קלט תקין יהיה לדוגמה:

- אני לומדת בשמחה
- אפרת גרה בחיפה

קלט לא תקין יהיה לדוגמה:

- אני לומדת JavaScript - יש כאן טקסט שאיננו בעברית
- אפרת גרה בפתח תקווה - כאן יש לנו 4 מילים במקום 3
- נועה גרה בחיפה - המילה הראשונה לא מתחילה באות א'

וכן הלאה.

ביטוי רגולרי הוא משהו מאד דומה לתבניות שהגדרנו למעלה. ההבדל הוא שבמקום לכתוב בעברית "תבנית כזו וכזו" - אנחנו מגדירים את התבנית באמצעות מילות/תווי מפתח. כך שכל תו מגדיר חלק מסויים מהקלט. באופן הזה המחשב מסוגל לפענח האם קלט מסויים תואם או לא תואם לתבנית שהגדרנו.

נתחיל מהבסיס

ביטוי רגולרי יתחיל ויסתיים בתו / (סלאש, נמצא בכפתור של "חילוק" במקלדת המספרים). בין סלאש אחד לשני נוכל לכתוב את התבנית.

ברוב הדוגמאות שבהמשך המדריך לא נכתוב את הסלאשים כדי להציג את התבניות בצורה פשוטה וקריאה יותר, אבל בכל אחת מהתבניות הסלאשים אמורים להופיע.

לדוגמה:

/me/

התבנית מחפשת בתוך הקלט את האות m ומיד אחריה e.

שימי ❤️ שהיא תחפש את התווים הללו גם כחלק ממילים אחרות.

דוגמאות לקלט תקין:

- Give **me** some code.
- Did you call **me**?
- Co**me** here

אבל קלט לא תקין יהיה לדוגמה:

- Hello world!
- Members list

אמנם בדוגמה השניה התווים me מופיעים, אבל האות m מופיעה כאות גדולה ולכן לא תואמת את התבנית שהגדרנו.

עבודה עם RegEx בשפת JS

כמו שנאמר למעלה, ביטויים רגולריים - RegEx - הם מושג כללי בעולם התכנות, ולא שייכים דווקא לשפת JavaScript.

בפרק הזה נכיר פונקציות JS לעבודה עם ביטויים רגולריים, ולאחר מכן נראה דוגמאות שונות לשימוש בהן בפרויקטים שונים.

RegExp

שפת JS הגדירה אובייקט ייעודי לעבודה עם ביטויים רגולריים. האובייקט נקרא **RegExp**.

כדי להתחיל לעבוד עם ביטוי רגולרי עלינו ליצור מופע של הטיפוס הזה. אפשר ליצור את המופע בשני דרכים:

- **ביטוי רגולרי מילולי** (באנגלית נקרא literal):

```
const re = /ab+c/;
```

שימי ❤️ לא לעטוף את הביטוי במרכאות. JS יודעת לתרגם באופן אוטומטי את הקבוע הזה לאובייקט.

- **אובייקט עם פונקצית אתחול**:

```
const re = new RegExp("ab+c");
```

באפשרות הזו **לא** נכתוב את הסלאשים שצריכים להופיע בתחילת וסוף הביטוי. JS מוסיפה אותם באופן אוטומטי.

בשני המקרים התוצאה דומה, ונוכל להפעיל את אותן פונקציות על הקבוע שלנו.

אפשרות נוספת היא כלל לא להכניס את הביטוי לתוך קבוע, אלא ישירות לתוך הפונקציה הרצויה. נראה דוגמאות בהמשך.

test() - בדיקת התאמה

הפונקציה test היא פונקציה ששייכת לאובייקט RegExp, ותחזיר לנו true/false בהתאם לשאלה האם סטרינג מסויים תואם או לא תואם לביטוי הרגולרי.

לדוגמה בדיקה האם סטרינגים מכילים ספרות.

```
// יצירת אובייקט ביטוי רגולרי
const digitPattern = /\d/;
// יצירת מחרוזות
const text1 = "Hello123";
const text2 = "NoDigitsHere";
```


אובייקטים חשובים נוספים בJS

כאשר אנחנו נכנסות לעמוד, או מרעננות עמוד, שפת JS מייצרת לנו באופן אוטומטי מספר אובייקטים חשובים שבהם אנחנו יכולות להשתמש בקוד.

עד כה דיברנו על document, שהוא אובייקט שמכיל את כל העץ הHTML'י כאובייקטים שאליהם אפשר לגשת דרך הקוד, וכן פונקציות נוספות שקשורות לHTML.

חשוב לדעת שdocument הוא רק חלק אחד קטן בתוך אובייקט גדול יותר, שנקרא window. וכן ישנם אובייקטים נוספים שכדאי להכיר.

האובייקט window

האובייקט window מייצג חלון שפתוח בדפדפן, ומאפשר לבצע פעולות שקשורות בחלון של הדפדפן, בקישור, פתיחת חלון חדש, הדפסה ועוד.

כרגיל, אנחנו לא נלמד את כל הפונקציות הקיימות, אלא נתמקד ביותר שימושיות.

print - הדפסה

התחביר:

```
window.print();
```

הרצה של הפקודה הזו בJS תפתח חלונית להדפסת העמוד ושמירה שלו לPDF. החלונית שנפתחת היא ברירת מחדל של הדפדפן שממנו גולשים, ואין אפשרות לשלוט עליה יותר מידי.

שימי ❤️שהרבה פעמים העמוד נראה על הפנים בהדפסה. כדי להוסיף CSS ייעודי למצב הדפסה - נכתוב את הקוד בתוך @media באופן הבא, בדומה לאיך שעובדים עם @media ברספונסיביות:

```
@media print{
  a{
    display: none;
  }
}
```

בדוגמה הזו, במצב הדפסה יוסתרו כל הקישורים.

בגלל שאי אפשר לדבג את חלונית ההדפסה, ולפעמים נרצה לבדוק רכיבים בF12, כדי לתקן את הנראות שלהם בהדפסה - אפשר להגדיר בF12 אמולטור הדפסה. כלומר - הדמיה של מצב הדפסה. זה לא זהה אחד לאחד להדפסה אמיתית, אבל מספק תמונה די טובה של איך העמוד יראה בהדפסה, ומאפשר לעבוד עם F12 במצב הדפסה.

העברת נתונים בין עמודים

במקרים רבים עלינו להעביר נתונים מעמוד אחד לעמוד אחר, וJS מאפשרת לנו 2 דרכים לבצע את זה.

האפשרות הראשונה שאתה אנחנו כבר מכירות היא שמירה לlocalStorage. נתון ששמור באחסון המקומי נגיש מכל העמודים שבאתר שלנו. באופן הזה אפשר לשמור נתונים כמו סל הקניות של הגולש, מי הגולש שמחובר כרגע וכדו'. נשתמש באחסון המקומי כאשר יש נתון שרלוונטי ברמת כל האתר/אפליקציה.

אבל כאשר נדרשת העברת מידע רק מעמוד אחד לעמוד אחר - אין טעם לשמור אותו באחסון המקומי וזה גם לא נכון.

להלן מספר דוגמאות:

- במסך הראשי הגולש בוחר את רמת המשחק ואז לוחץ על הכפתור "התחל משחק". אני מעבירה אותו לעמוד של המשחק, אבל חייבת לדעת באיזו רמה הוא בחר לשחק. זה רלוונטי רק עכשיו ורק למשחק הזה.
- הגולש מילא את פרטיו בטופס תרומה באתר. לאחר מכן אני מעבירה אותו לעמוד חדש של "תודה על תרומתך" וצריכה לשלב נתונים שהוא הזין בטופס, על מנת לנסח את ההודעה בצורה אישית יותר. "משה, תודה על תרומתך בסך 140 ש"ח". גם כאן הנתונים רלוונטיים לי רק עכשיו ורק בעמוד הזה.
- כאשר מחפשים משהו בטופס החיפוש באתר מועברים לעמוד תוצאות חיפוש. בעמוד תוצאות חיפוש אני צריכה לדעת את אילו תוצאות עלי לשלוף מהDB כדי להציג בעמוד על סמך מילת החיפוש שהגולש הזין בעמוד הקודם. זה רלוונטי רק הרגע ולא באף מקום או זמן אחר באתר.

בדוגמאות אלו ובמקרים רבים נוספים שמירה לאחסון המקומי היא פשוט לא נכונה. אני לא רוצה לשמור את הפרטים וסכום התרומה שהגולש הזין באחסון המקומי, כי זה רלוונטי לי רק בעמוד הנוכחי שאליו הגולש מועבר מיידית, ולא רלוונטי לשאר הגלישה שלו באתר.

במקרים הללו נשתמש באפשרות השניה של העברת נתונים - העברת נתונים כפרמטרים בקישור.

מה הם פרמטרים בקישור?

קישור של עמוד מורכב מכמה חלקים. לדוגמה:

https://my-site.com/thanks/?sum=140&name=Moshe

תחילה יש לנו את ה**פורט** - https, אחר כך השם של האתר, ה**דומיין** - my-site.com.

אחר כך יכול להופיע ניתוב פנימי, במקרה הזה **thanks**, זה הנתיב בתוך האתר לעמוד הספציפי שבו אני גולשת כרגע. הנתיב יכול להיות ארוך יותר. הנתיב הפנימי זה החל מהדומיין ועד סימן השאלה.

החל מהתו **?** מתחילים פרמטרים שמשורשרים לקישור. אלו נתונים שהועברו מהעמוד הקודם לעמוד הזה.


```
const username = paramsObj.get("name");
const sum = paramsObj.get(“sum”);
const str = `${username}, thank you for your donation of
${sum}`;
document.querySelector(“#message”).innerText = str;
```

דוגמאות נוספות

כאמור, ישנם מגוון שימושים אפשריים לפרמטרים.

במקרה שלמעלה שלפנו את הפרטים שהגולש הזין בטופס תרומה כדי להציג לו הודעה מותאמת.

אם אנחנו ב**משחק הטריוויה** שהדגמנו למעלה גם כן, נשלוף את מזהה הקטגוריה מהקישור, ואז נשתמש בפונקציה filter על מערך השאלות, כדי לקבל אך ורק שאלות שקשורות לקטגוריה הרלוונטית, ואז נוכל לצייר השאלות לגולש.

כנ"ל באתר חנות. הגולש לחץ על הקטגוריה "נעליים" ועבר לעמוד products.html?cat=20. בעמוד מוצרים נשלוף את הפרמטר cat, נסנן את מערך המוצרים כך שיציג רק את המוצרים שמשוייכים לקטגוריה מספר 20, ואז נרוץ עליהם ונצייר.

כל קובית מוצר היא קישור לעמוד מוצר. וגם כאן התהליך חוזר על עצמו. יש לנו רק עמוד אחד שמציג מוצר בודד, נניח שקוראים לו single-product.html.

בעמוד הקטגוריה, כאשר מציירים את קוביות המוצרים, נדאג שהקישור יכיל את מזהה המוצר, לדוגמה:

single-product.html?prod=12

בעמוד מוצר תחילה נשלוף את מזהה המוצר מתוך הקישור, ואז נשתמש בפונקציה find כדי לשלוף את המוצר שהמזהה שלו הוא 12, מתוך מערך המוצרים. אז נוכל בקלות להציג את המוצר על המסך.

כנ"ל במשחקים שונים. הגולש בחר רמה "easy" בשדה level שבטופס במסך הראשי. באמצעות action של הטופס העברנו את הגולש לעמוד המשחק ושם נשלוף את הפרמטר "level" מתוך הקישור. כעת יש לי ביד את הבחירה של הגולש ואני יכולה להפעיל את המשחק בהתאם לרמה שהוא בחר.

נספח

איתור באגים ופתרון תקלות

שגיאות נפוצות ואיך לפתור אותן

עבודה עם F12, נקודות עצירה ו-debugger

שגיאות נפוצות ואיך לפתור אותן

בפרק הזה נראה כמה מהשגיאות היותר מוכרות בJS, נבין מה גורם להן ואיך לפתור אותן.

חשוב לזכור שהבאנו כאן מדגם של שגיאות נפוצות. לא את כל השגיאות ולא את כל הסיטואציות האפשריות.

בכל מקרה שבו משהו בקוד לא עובד כמצופה - נפתח את הקונסול ונבדוק האם קיימות שגיאות. במידה וכן - נוכל לראות בקונסול בצורה ברורה למדי מה השגיאה ומאיזו שורה באיזה קובץ היא מגיעה. בדיקה זריזה של השגיאה והמקור שלה יכוונו אותך בעז"ה גם לפתרון.

Uncaught ReferenceError: … is not defined

השגיאה ReferenceError תופיע כאשר אנחנו מנסות לקרוא למשתנה או פונקציה שלא קיימים.

לדוגמה:

```
console.log(fakeVar); // Uncaught ReferenceError: fakeVar is not
defined
myfunc(); // Uncaught ReferenceError: myfunc is not defined
```

<u>איך נתקן את השגיאה</u>

1. ודאי שקראת למשתנה לאחר שהצהרת עליו.
2. ודאי שאין שגיאת כתיב בשם הפונקציה/משתנה.

Uncaught TypeError: Cannot set property '...' of null or undefined

השגיאה מתריעה לנו שאנו מנסות להפעיל פונקציה או לקרוא למאפיינים של מישהו שלא קיים או שהוא ריק.

השגיאה נפוצה בעיקר בעת הרצת פונקציה/קריאה למאפיין של אובייקט, וכן נפוצה מאד בעת הפעלת מניפולציות/קריאה למאפיינים/הצמדת אירועים לאלמנטים שלא קיימים בDOM.

לדוגמה:

```
let myVar;
myVar.toUpperCase();
// Uncaught TypeError: Cannot read properties of undefined
(reading 'toUpperCase')
const element = document.querySelector("#someID");
element.classList.add("hello");
// Uncaught TypeError: Cannot read properties of null (reading
'classList')
```

תוכן העניינים

