עד עכשיו עשינו היכרות עם HTML ו-CSS. שתי השפות, אפשר לומר שהן יותר קובצי הגדרות מאשר שפת תכנות במובן המקובל - למרות שיהיו שיתווכחו על העניין. היום נכיר את Javascript.

ג'אווה סקריפט היא שפת תכנות במלוא מובן המילה והיא שפה שהדפדפן יודע להריץ.

אז כמו לכל שפת תכנות גם לג'אווה סקריפט יש משתנים, אופרטורים, יכולת לבצע חישובים, פונקציות, מחלקות וכן הלאה.

**הערות**

לשפת תכנות יש צורת כתיבה מסויימת ומילים שמורות. כל שורת קוד שכותבים בקובץ צריכה להיכתב על-פי הכללים הללו. בכל אופן, יש אפשרות לכתוב הערות בקוד כשמפרש הקוד - במקרה שלנו הדפדפן - יתעלם מהם וימשיך הלאה. ההערות נועדו לתת מידע נוסף למתכנת. סיבה נוספת להשתמש בהערות היא במידה ונרצה להשאיר קטע קוד מסוים בקובץ אך נרצה שהמפרש ידלג על הקטע קוד הזה ולא יבצע אותו. נראה שימוש לכך בהמשך.

[מעבדה: הערות]

יש אפשרות לכתוב הערה בשורה אחת ועושים זאת ע"י כתיבת פעמיים סלאש ואת הטקסט הרצוי מיד לאחר מכן. דוגמא:

// I am a single line comment

יש גם אפשרות לכתוב בלוק הערות שיתפרש על יותר משורה אחת. דוגמא:

/\*

I am

a multi line comment

\*/

בדרך כלל, כל עורך קוד שתעבדו איתו, יצבע את הטקסט שנמצא בהערה בצבע אחיד, לרוב בצבע ירוק.

שימו לב לדרך נוחה לכתיבת קוד אותו נרצה לעתים לבצע ולעתים נרצה שמפרש הקוד יתעלם ממנו.

בצורה הבאה הקוד יתבצע:

/\*\*/

console.log('Experimental code');

/\*\*/

ולאחר שהוספתי תו רווח בין הכוכבית השניה לסלאש בשורה הראשונה, הקוד לא יתבצע:

/\*\* /

console.log('Experimental code');

/\*\*/

**משתנים**

מה זה משתנה? משתנה הוא מעין תא לאחסון של מידע. המידע יכול להיות מספר שלם, שבר עשרוני, מחרוזת טקסט וכדומה. בהמשך נכיר סוגים נוספים. מה שחשוב לדעת כרגע זה שבניגוד לשפות אחרות, בג'אווה סקריפט אין צורך להגדיר את סוג המשתנה. אגב, יש שפה שנכתבה ע"י מיקרוסופט - היא נקראת TypeScript - והיא שפה כמעט זהה ל-Javascript כשאחד העקרונות המרכזיים שלה שהיא כן מתייחסת לסוג המשתנה. בכל מקרה, בסופו של דבר ממירים את הקוד שלה (בצורה אוטומטית, כמובן) לג'אווה סקריפט - משום שזאת השפה שהדפדפן מכיר.

[מעבדה: משתנים]

1. צרו תיקיה בשם **lab-5** ובתוכה קובץ **index.html** וקובץ נוסף **variables.js**.
2. בקובץ index.html צרו **שלד** **לדף** **HTML5**.
3. לפני התגית הסוגרת של body הוסיפו תגית שנקראת script. הוסיפו מאפיין (attribute) שנקרא src. אם אתם משתמשים ב-VS Code הוא יציע **script:src** בחרו באפשרות הזאת. בערך של src הוסיפו את הקובץ **variables.js**.
4. כתבו ב-app.js את הקוד הבא:

myVar = 5;

1. שימו לב לדרך שבה כתבתי את שם המשתנה המורכב משתי מילים מחוברות. המילה הראשונה כ-lower-case וכל מילה שבאה לאחר מכן, upper-case. הדרך הזאת נקראת **camel-case**. אין הכרח לכתוב משתנים בצורה הזאת בג'אווה סקריפט אך זה הנוהג.
2. שמרו את הקובץ ובדקו בדפדפן מה קיבלתם? כלום, דף ריק. המשתנה קיבל את הערך 5, הוא מאוחסן בזיכרון של המחשב אבל אנחנו לא עושים איתו שום דבר מיוחד.
3. הפעולה הבסיסית שאפשר לעשות עם משתנים הוא להדפיס אותם ב-console שנמצא ב-DevTools.
4. הוסיפו את השורה הבאה:

console.log('myVar = ', myVar);

1. שמרו, עברו לדפדפן, פתחו את ה-DevTools ורפרשו את הדף. בדקו בלשונית console את ה-output.
2. אגב, שימו לב שיש שתי לשוניות של console שאפשר לבחור - לפי הצורך.
3. הוסיפו את שורות הקוד הבאות:

myVar = 6 + 2.23;

console.log('myVar = ', myVar);

1. שמרו ובדקו ב-console.
2. שימו לב שביצענו פעולת חיבור בין המספרים. באותה צורה ניתן לבצע פעולות חשבון נוספות, כמובן.
3. הוסיפו את שורות הקוד הבאות:

myVar = 'just a simple string';

console.log('myVar = ', myVar);

1. מה שראינו בינתיים שבפעולה הראשונה עשינו השמה של מספר שלם למשתנה myVar, לאחר מכן לאותו משתנה עשינו השמה של פעולת חיבור בין מספר שלם לשבר עשרוני ולאחר מכן עשינו השמה של מחרוזת לאותו משתנה. הדגמנו שבג'אווה סקריפט אין התייחסות לסוג המשתנה.
2. יש שלוש דרכים להגדיר משתנים בג'אווה סקריפט. עושים זאת באמצעות הוספת מילה מסויימת לפני שם המשתנה.
3. הדרך הראשונה היא באמצעות המילה var והיא זהה לדרך של להגדיר משתנה ללא הוספת מילה מסויימת לפני שם המשתנה - כפי שראינו קודם לכן. הוסיפו את המילה var לפני המשתנה myVar בשלוש הדוגמאות הקודמות. שמרו ובדקו ב-console.

// the "var" keyword

var myVar = 5;

console.log('myVar = ', myVar);

var myVar = 6 + 2.23;

console.log('myVar = ', myVar);

var myVar = 'just a simple string';

console.log('myVar = ', myVar);

1. במשך שנים זאת הייתה הדרך היחידה להגדיר משתנים - עם המילה var - כאשר בכל מצב ניתן להגדיר את המשתנה מחדש.
2. בשנים האחרונות נוספו שתי דרכים חדשות ליצירת משתנים וחשוב להכיר אותן. מאז שהן נוספו הן תפסו את מקומה של var.
3. לפני שנעבור לדרך הבאה ליצירת משתנה הוסיפו הערה לפני קטע הקוד שכתבתם וסגרו את כל קטע הקוד בהערה לבלוק מהסוג האחרון שלמדנו, כך שיהיה ניתן בקלות לאפשר את ביצוע הקוד ע"י מחיקת תו הרווח שבין הכוכבית לסלאש. בצורה הבאה:

/\*\* /

// the "var" keyword

var myVar = 5;

console.log('myVar = ', myVar);

var myVar = 6 + 2.23;

console.log('myVar = ', myVar);

var myVar = 'just a simple string';

console.log('myVar = ', myVar);

/\*\*/

1. הדרך הבאה ליצירת משתנה יוצרת משתנה אותו ניתן לדרוס ומשתמשים במילה let כדי ליצור אותו.
2. בשורה חדשה כתבו את הקוד הבא:

// the "let" keyword

let myOtherVar = 5;

console.log('myOtherVar = ', myOtherVar);

let myOtherVar = 1.5;

console.log('myOtherVar = ', myOtherVar);

1. שמרו ובדקו בדפדפן. קיבלנו שגיאה:

Uncaught SyntaxError: Identifier 'myVar' has already been declared

1. השגיאה אומרת ש-myVar הוגדר כבר קודם ואסור להגדיר משתנה יותר מפעם אחת.
2. שימו לב שבנוסף להודעת שגיאה, אף פקודה אחרת לא התבצעה. זאת בניגוד למקרים אחרים. בואו נראה דוגמא.
3. מחקו את המילה let מהשורה האחרונה בה נעשית הגדרה למשתנה ומחקו את האות m, לצורך העניין, מ-myOtherVar בשורה של ה-console.log. השורה האחרונה שלנו.

let myOtherVar = 5;

console.log('myOtherVar = ', myOtherVar);

myOtherVar = 1.5;

console.log('myOtherVar = ', yOtherVar);

1. שמרו ובדקו ב-console. הקוד התבצע ברובו, אך נפל בשורה האחרונה עם השגיאה:

Uncaught ReferenceError: yOtherVar is not defined at app.js:[line-number]

1. השגיאה אומרת שהמשתנה שביקשנו להדפיס אינו מוגדר.
2. הסיבה לכך שהבאג הקודם שבו הגדרנו בעזרת let את אותו משתנה גרם לכך שלא תתבצע אף שורה היא שהדפדפן קורא את הקוד ומסדר אותו מחדש בזיכרון כך שהגדרת משתנים מתבצעת לפני פעולות אחרות - למרות העובדה שכתבנו את הגדרת המשתנה לאחר פעולות אחרות - במקרה שלנו console.log.
3. אגב, סידור הקוד מחדש ע"י הדפדפן נקרא הוייסטינג (hoisting, הנפה מלשון מנוף).
4. חזרו לקוד והחזירו את האות m אותה מחקנו מ-myOtherVar בשורה האחרונה. שמרו ובדקו בדפדפן. עכשיו כל הקוד בוצע בהצלחה.
5. לסיכום, בעזרת המילה let אנחנו מגדירים משתנה אותו ניתן לערוך ולשנות בהמשך אך לא ניתן להגדיר אותו מחדש.
6. עטפו את כל קטע הקוד בהערה של בלוק כך שיהיה ניתן לחזור לקוד מאוחר יותר ולשחק איתו.
7. הדרך השלישית ליצירת משתנה נעשית בשימוש במילה const והיא יוצרת משתנה שלא ניתן לערוך אותו. הוא משתנה מסוג קבוע.
8. בשורה חדשה כתבו:

// the "const" keyword

const myConst = 'I am a const';

console.log('myConst', myConst);

1. שמרו ובדקו ב-console. נוצר משתנה עם המחרוזת שכתבתם והיא מודפסת ב-console.
2. שנו את תוכן המשתנה, למשל לדוגמא הבאה:

myConst = 123;

console.log('myConst', myConst);

1. בדקו ב-console. רוב הקוד התבצע וקיבלנו שגיאה:

Uncaught TypeError: Assignment to constant variable. at app.js:[line-number]

1. ניסינו לבצע השמה לקבוע. הקוד שנמצא בשורות הקודמות התבצע משום שלא מדובר על שגיאה ברמת הגדרת משתנה אלא השמה.
2. יש תכונות נוספות למשתנים מהסוגים השונים, נגלה אותם בהמשך.
3. הדבר הבא שנעשה זה לראות איך משלבים משתנים ומחרוזות טקסט.
4. כתבו את הקוד הבא:

// Variables and strings

let followersLabel = 'followers';

let followersNum = 8;

console.log('There are ' + followersNum + ' ' + followersLabel);

1. שמרו ובדקו בדפדפן. חיברנו טקסט ומשתנים. שימו לב לסימן פלוס שבו השתמשנו. זה לא ממש נוח לכתוב בצורה הזאת ולכן לפני מספר שנים נכנסה לשימוש צורת כתיבה נוספת.
2. הוסיפו את הקוד הבא:

console.log(`There are ${followersNum} ${followersLabel}`);

1. שמרו ובדקו. צורת כתיבה נוחה יותר. בכל אופן, יש אפשרות להשתמש בשתי הצורות.

**משתנים בוליאנים, אופרטורים לוגיים, אופרטורים להשוואה והתניות**

כמו ברוב השפות, גם בג'אווה סקריפט יש אפשרות ליצור התניות ובהתאם לתוצאה - אמת או שקר (true/false) - לבצע או לא לבצע פעולות מסוימות. משתנה שערכו הוא true או false נקרא משתנה בוליאני.

[מעבדה: משתנים בוליאנים ואופרטורים לוגיים]

1. נתחיל בהיכרות עם משתנים בוליאנים. קודם לכן ביצענו פעולת חשבון שכללה את האופרטור פלוס לצורך חיבור בין שני מספרים. יש, כמובן, אופרטורים נוספים לצורך ביצוע פעולות חשבון - כמו חיסור, כפל, חילוק, חזקה, שארית וכן הלאה. לצורך ביצוע פעולות לוגיות אנו משתמשים באופרטורים לוגיים. יש שלושה אופרטורים כאלה: OR, AND ו-NOT.
2. נגדיר שני קבועים, הראשון יהיה true והשני false:

const a = true;

const b = false;

1. נתחיל ב-NOT. האופרטור הזה הוא מהפך. true הופך ל-false ולהיפך. אנו משתמשים בסימן קריאה על מנת לציין את האופרטור NOT. כתבו בשורה חדשה:

console.log('NOT', !a);

1. ערך המשתנה a שערכו היה true השתנה ל-false.
2. נעבור ל-OR. אנו משתמשים בשני קווים (pipes). הכלל של "או" קובע שאם לפחות אחד המשתנים הוא true - התוצאה כולה תהיה true. כתבו:

console.log('OR', a || b);

1. קיבלנו true.
2. את האופרטור AND מציינים בעזרת שני סימני and: &&. הכלל של "וגם" קובע שכל המשתנים חייבים להיות true כדי שהתוצאה הסופית תהיה true - אחרת false. כתבו:

console.log('AND', a && b);

1. קיבלנו false.

[מעבדה: אופרטורים להשוואה]

1. יש אפשרות להשוות בין משתנים כשהתוצאה תהיה true או false. הדבר נעשה בעזרת הסימנים: שווה, גדול מ… וקטן מ… ושילובים ביניהם.
2. נגדיר מספר משתנים איתם נעבוד בהמשך:

const a = 2;

const b = 4;

const c = 2;

const d = '4';

1. כתבו את השורה הבאה:

console.log(a > b);

(false)

1. ההשוואה הבאה:

console.log(a >= c);

(true)

1. ההשוואה הבאה:

console.log(b == d);

(true)

1. כאשר משתמשים בפעמיים הסימן "שווה" (=), אין התייחסות לסוג המידע ומיקומו בזיכרון - לכן ההשוואה בין הספרה 4 לתו 4 תחזיר true. כדי להשוות האם המשתנה זהה מבחינת מיקומו בזיכרון כך שגם סוג המידע זהה (by reference) אנו משתמשים ב-3 פעמים הסימן שווה:

console.log(b === d);

(false)

[מעבדה: התניות]

1. נשתמש בקבועים מהמעבדה הקודמת.
2. הדרך הראשונה ליצור משפט התניה היא באמצעות המילה if. בצורה הבאה:

if (a < b) {

console.log('a is less than b');

}

1. יש אפשרות ליצור בלוק פעולה במידה ובתוצאה של if קיבלנו false. לצורך כך אנו משתמשים ב-else:

if (a > b) {

console.log('a is greater than b');

}

else {

console.log('a is less than b');

}

1. אפשרות נוספת ליצור משפט התניה היא בעזרת סימנים וללא מילים כך שסימן שאלה (?) מייצג if והסימן נקודותיים (:) מסמן else. בצורה הבאה:

const result = (a < b) ? 'a is less than b' : 'a is greater than b';

console.log(result);

1. דרך נוספת להתניה היא בעזרת switch ו-case.
2. ניצור את הקבועים הבאים ונחליף ביניהם בכל דוגמא כשאת השורות בהן לא נשתמש נתחום בהערה:

const result = '111';

// const result = '222';

// const result = '333';

// const result = 'abcd';

1. משפט של switch case כותבים בצורה הבאה:

switch (result) {

case '111':

console.log('result is 222');

break;

case '222':

console.log('result is 222');

break;

case '333':

default:

console.log('I am default');

break;

}

1. שימו לב שיש case למצבים שונים ובמידה ואין אחד ה-case-ים הללו, יתבצע הקוד שב-default.
2. התוצאה הראשונה של 111 היא ה-case הראשון, של 222 היא ה-case השני. של 333 היא אותו case של ה-default משום שאין break. והתוצאה של abcd גם היא אותה תוצאה של ה-default.

**לולאות**

יש אפשרות להריץ קטע קוד מספר רב של פעמים לפי הצורך. יש מספר דרכים לעשות את זה.

[מעבדה: לולאות]

1. לולאת while:

let x = 0;

while (x < 10) {

console.log(x);

x++;

}

1. לולאת do while:

let x = 0;

do {

console.log(x);

x++;

} while (x < 10);

1. לולאת for:

for (let i = 0; i < 10; i++) {

console.log(x);

}

1. יש לולאות נוספות, נכיר אותן בהמשך כשנעסוק עם מערכים ואובייקטים.

**פונקציות**

פונקציות מאפשרות לתחום קטע קוד שמטרתו לבצע פעולה מסוימת ולאפשר ביצוע של הקוד לפי הצורך. לעתים הפונקציה אך ורק תבצע פעולה מסויימת ולעתים בנוסף היא תחזיר ערך. נראה בהמשך דוגמאות לכך.

[מעבדה: פונקציות]

1. צרו קובץ חדש בשם functions.js וכתבו לינק שיוביל אליו בקובץ ב-HTML.
2. יש שתי דרכים עיקריות ליצירת פונקציות בג'אווה סקריפט, הראשונה מאוד ותיקה והשניה חדשה יחסית - מהשנים האחרונות. סביר להניח שתתקלו בשתי הדרכים לכן נכיר את שתיהן.
3. הדרך הראשונה, כתבו את המילה function ואחריה רוחח ואת שם הפונקציה - למשל, myFunc. לאחר מכן סוגריים מעוגלים ולבסוף סוגריים מסולסלים.

function myFunc() {

console.log('I am a function');

}

1. מה שעשינו זה להגדיר את הפונקציה. עד שלא נקרא לה, הקוד בתוכה לא יתבצע.
2. על מנת לקרוא לפונקציה ולבצע את הקוד שכתבנו בתוכה, כתבו:

myFunc();

1. הפונקציה הפשוטה שכתבנו מבצעת פעולה מסוימת ולא מחזירה ערך מסויים.
2. דוגמא לפונקציה שמחזירה ערך:

function myOtherFunc() {

return 'I am a function';

}

1. נקרא לפונקציה ואת הערך שהיא מחזירה נאחסן במשתנה ונדפיס אותו ל-console:

const myFuncResponse = myFunc();

console.log(myFuncResponse);

1. לפונקציה יש אפשרות להעביר פרמטרים. כתבו את הפונקציה הבאה:

function addition(x, y) {

return x + y;

}

1. הגדירו את המשתנים הבאים:

const x = 3;

const y = 5;

1. קראו לפונקציה עם הפרמטרים שהגדרנו והדפיסו את התוצאה:

const addResult = addition(x, y);

console.log(`The result of ${x} + ${y} is: ${addResult}`);

1. עד כאן לדרך הקלאסית לכתיבת פונקציות. נעבור לדרך החדשה יחסית. היא נקראת arrow function. הדרך לכתוב אותה היא בצורה הבאה. כתבו את המילה const מיד לאחר מכן את שם הפונקציה, אחר כך שווה, סוגריים מעוגלים לפרמטרים, חץ שמורכב מהסימן שווה מחובר לסימן קטן מ... ואז סוגריים מסולסלים.

const myFunc = () => {

return 'I am an arrow function';

}

1. נקרא לפונקציה כך שתחזיר את הערך שלה לתוך משתנה - ונדפיס את המשתנה:

const myFuncResponse = myFunc();

console.log(myFuncResponse);

מעתה ואילך רוב הפונקציות שאכתוב יהיו בצורת הפונקצית חץ.

**מחלקות - classes**

מחלקה מגדירה עצם מסוים - אובייקט. אגב, אם נתקלתם במונח object oriented language או בעברית שפה מונחית עצמים - הכוונה היא בדיוק לנושא שנלמד בחלק הזה.

ניקח לדוגמא, משתמש - user. למשתמש יש תכונות, כמו למשל: שם פרטי, שם משפחה, מספר טלפון וכולי. הם בעצם משתנים. בנוסף על כך, יש אפשרות לבצע פעולות שקשורות למשתמש. פונקציות.

בעולם המחלקות, המשתנים נקראים properties והפונקציות נקראות methods.

נעבור לדוגמא בקוד.

[מעבדה: מחלקות]

1. ליצירת מחלקה בג'אווה סקריפט כתבו את המילה class, את שם המחלקה וסוגריים מסולסלים. נהוג להתחיל את שם המחלקה עם אות גדולה.

class User {

}

1. את התכונות - ה-properties - מגדירים בד"כ בפונקציה מובנית שנקראת constructor. עושים זאת בצורה הבאה:

constructor() {

this.firstName = null;

this.lastName = null;

}

1. יש מספר דרכים להגדיר מתודות כך שיתנהגו בצורה שונה. כרגע, אני לא רוצה להעמיס בפרטים ולכן נראה את הצורה הפשוטה ובצורה הזאת, היא מוגדרת כפונקציה לכל דבר, בצורה הבאה:

returnFullName() {

return `${this.firstName} ${this.lastName}`;

}

1. יצרנו מחלקה. המחלקה היא מעין blue-print ליצירת אובייקטים על-פי ההוראות שכתובות בתוכה. נשתמש בה בצורה הבאה:

const userExample = new User();

1. בהמשך נציב ערכים ל-properties של השם הפרטי ושם המשפחה בצורה הבאה:

userExample.firstName = 'Yaniv';

userExample.lastName = 'Or';

1. באותה דרך נקרא למתודה היחידה שהגדרנו ונבצע את הקוד שנמצא בתוכה. את הערך נחזיר למשתנה:

const fullName = userExample.returnFullName();

console.log(fullName);

1. שימו לב שה-constructor היא פונקציה וכמו לכל פונקציה יש אפשרות להעביר לו פרמטרים. אז במקום שנציב את השם הפרטי ושם המשפחה כפי שעשינו קודם לכן, נעביר את הערכים ישירות ל-constructor בזמן יצירת האובייקט החדש.
2. נבצע מספר תיקונים לקוד. דבר ראשון נאפשר העברת פרמטרים ל-constructor ונציב אותם ב-properties. בצורה הבאה:

constructor(firstName, lastName) {

this.firstName = firstName;

this.lastName = lastName;

}

1. התיקון הבא שנעשה זה להציב ערכים ישירות ביצירת האובייקט, בצורה הבאה:

const userExample = new User('Yaniv', 'Or');

1. כמובן, אין צורך יותר בהצבת הערכים ל-properties, מחקו את השורות הבאות:

userExample.firstName = 'Yaniv';

userExample.lastName = 'Or';

יש עוד הרבה מאוד חומר ללימוד בנושא המחלקות, כרגע זה מספיק בשבילנו.

**סוגי משתנים**

נחזור למשתנים. השתמשנו עד כה במשתנים שמחזיקים שני סוגי מידע עיקריים: מספרים ומחרוזות.

נכיר שני סוגים חשובים נוספים.

הראשון הוא **מערך**. מערך הוא משתנה שמכיל בתוכו יותר מערך אחד. בג'אווה סקריפט, מערך יכול להחזיק ערכים מסוגים שונים. דבר נוסף חשוב לגבי מערך הוא שיש חשיבות לסדר של הערכים.

[מעבדה: מערכים]

1. בתור התחלה נכתוב מערך של מחרוזות, לצורך העניין רשימת פירות. כתבו את המילה const מיד אחריה את שם המערך, במקרה שלנו fruits את הסימן שווה ולאחר מכן סוגריים מרובעים ונקודה פסיק. בצורה הבאה:

const fruits = [

];

1. את שמות הפירות נכתוב בתוך גרש בודד או גרשיים, מופרדים בפסיקים:

const fruits = [

'apple',

'orange',

'banana',

'mango',

];

1. יש אפשרות להדפיס ל-console את המערך כולו. בצורה הבאה:

console.log('fruits:', fruits);

1. כפי שציינתי, במערך - הסדר של הערכים חשוב ולכל ערך יש מיקום במערך - index. חשוב מאוד: ה-index של הערך הראשון הוא 0 - לא 1. נראה את הדבר בדוגמא הבאה:

console.log('The fruit in index 0 is:', fruits[0]);

1. דוגמא נוספת, הפרי הרביעי נמצא ב-index השלישי:

console.log('The fruit in index 3 is:', fruits[3]);

1. וכן הלאה. שימו לב למקרה בו אין ערך ב-index מסויים:

console.log('The fruit in index 12 is:', fruits[12]);

The fruit in index 12 is: undefined

1. עובדה נוספת אותה ציינתי ונדגים כעת היא שמערך יכול להכיל ערכים מסוגים שונים.

const mix = [

'apple',

2,

45.6,

() => { console.log(`I'm a function`)},

];

1. שימו לב לפונקציה שכתבתי בתור הערך הרביעי, ב-index מספר 3. גם פונקציה היא סוג של ערך. נדפיס את הערך של אינדקס 3:

console.log('The value in index 3 is:',mix[3]);

1. שימו לב שהפונקציה לא בוצעה. כדי לקרוא לפונקציה ולבצע את הפקודות שכתובות בתוכה נוסיף סוגריים מעוגלים. בצורה הבאה:

mix[3]();

1. אגב, פונקציה מהסוג הזה שאין לה שם והיא איננה מוגדרת בתוך משתנה נקראת פונקציה אנונימית.
2. לצורך ההדגמה, נעתיק את הפונקציה וניצור אותה עם שם מחוץ למערך ולפני הגדרת המערך, זה חשוב. כמו כן, נתקן את ה-index ה-3. הקוד המלא נראה כך:

const myFunc = () => console.log(`I'm a function`);

const mix = [

'apple',

2,

45.6,

myFunc,

];

console.log('The value in index 3 is:',mix[3]);

mix[3]();

1. עד כאן סוגי ערכים שונים באותו מערך. נחזור למערך של הפירות ונעבוד איתו.
2. יש בג'אווה סקריפט מספר פונקציות מובנות לעבודה עם מערכים.

5. להדגים const ו-let.