JAVASCRIPT

Objects Array & Functions

הדברים עליהם נעבור בשיעור

- Const let var
- json
- Data Types
- סוגי מפתחות שונים בתוך אובייקטים
- ננסה להבין את המשפט הבא:
- `const ROOT = document.getElementById("root")`
- `document.getElementById("root").addEventListener(“click”, ()=> 5);`

משתנים וקבועים ב - JAVASCRIPT

- ישנם שלושה סוגי מיכלים ב – javascript
  - var – השיטה הישנה ובה ניתן לשנות את סוג המידע לאחר קביעת ערכו הראשוני

```
var test = 5
test = "value"
```

  - let – הדרך המודרנית להגדרת משתנה. גם את סוג המידע בערך של מיכל זה ניתן לשנות

```
let test = 5
test = "value"
```

  - const – הגדרה של קבוע. המשתמעות היא שלא ניתן לשנות את סוג הערך שלו

```
const test = 5
```

מחרוזת תווים - string

- ישנם שלוש דרכים ליצור מחרוזת תווים:
  - עם מירכאות מכופלות " "
  - עם מירכאות לא מכופלות ' '
  - עם backtick ``

Object Syntax

- באובייקט אנחנו נזהה את המפתח ואת הערך על ידי מיקומם ביחס לנקודתיים
- אם מחרוזת התווים נמצאת מצד שמאל לנקודתיים זהו מפתח
- מה שנמצא מצד ימין לנקודתיים זהו הערך

```
{ key : value }
```

JSON

- הגדרה - JavaScript Object Notation
- קישור להגדרה בוויקיפדיה: https://he.wikipedia.org/wiki/JSON

```
[
  {
    "string": "string",
    "number": 5,
    "null": null,
    "object": {},
    "array": [],
    "Boolean": true
  }
]
```

Data Types in Javascript

- Number = 5, -5, 5.23, NaN, 0, …
- Boolean = true/ false
- String = “ ”, ‘ ‘, ` ${ }`
- Null – null
- Array = [ ]
- Undefined = undefined
- Object = { }, [ ], function(){}
- Function = ()=>{}

Objects

- כל אובייקט שאנו יוצרים נוצר בשבילו מקום הזיכרון
- באופן דיפולטיבי javascript נותנת לכל מחרוזת תווים אפשרית ערך דיפולטיבי של undefined
- כשאנחנו עושים השמה למפתח מסוים אנו למעשה דורסים את הערך הראשוני ש – javascript נתנה.

<u>דוגמה</u>:

```
const obj = {}
obj.key = undefined;
obj.key = "hallo";

console.log(obj.key);
```

- **האובייקט האחרון קובע!**

סוגריים מרובעים באובייקט

- המשמעות של סוגריים מרובעים בצד של המפתח (משמאל לנקודתיים) היא אזור של javascript בתוך האובייקט.
- אם מתקבלת בתוך הסוגריים המרובעים מחרוזת תווים הוא מתרגם אותה למפתח – { "key"] : "value" }
- אם הוא מזהה בתוך הסוגריים המרובעים פעולה חישובית, הוא יבצע אותה וימיר את התוצאה למחרוזת תווים - { [1 + 2] : "value" }
- אם הוא מזהה משתנה בתוך הסוגריים המרובעים הוא יחפש אותו מחוץ לאובייקט (לפני האובייקט):

Let key = "name";

{ [key] : "value" }

איך מזהים מפתח בתוך אובייקט

- הדרך לזהות מפתח בתוך אובייקט היא באמצעות הנקודה
- מה שנמצא מיד מצד שמאל לנקודה הוא האובייקט
- מה שנמצא מיד בצד ימין הוא המפתח

Document.getEelmentById

מחיקת מפתחות מתוך אובייקט

- הדרך למחוק מפתח וערך מתוך אובייקט היא באמצעות המילה השמורה delete
- <u>דוגמה</u>:

```
const obj = { name: “Yossi”}
delete obj.name
```

הוספה מפתחות לתוך אובייקט

- אם שם המפתח שאנו רוצים להכניס לאובייקט הוא שם תיקני לקיצור, ניתן להוסיף אותו באמצעות נקודה
- דוגמה:

```
const obj = {}
obj.name = "Yossi"
```

- אם השם אינו תקיני לקיצור דרך נוסיף אותו באמצעות סוגריים מרובעים

```
obj["my name"] = "david"
```

פונקציות באובייקט

- ה"שם המלא" של פונקציה בתוך אובייקט:

```
{
"fn" : function() {}
}
```

- קיצור הדרך אליו אנו מורגלים:

```
{
    fn() {}
}
```

ערכים בפונקציות

- כל פונקציה מחזירה ערך!!!!!
- מה יהיה הערך?
- הערך הדיפולטיבי הוא undefined
- אם רשום מה הפונקציה מחזירה בתוכה (return) אז היא תחזיר את מה שמוגדר לה.

הפעלת פונקציות שבתוך אובייקט

- מתכון הפונקציה fn בתוך האובייקט obj

```
const obj = {
    fn() {
        return 5
    }
}
```

העברת הערך של המפתח fn מתוך האובייקט obj ששווה למתכון של הפונקציה

```
obj.fn
```

הפעלת הפונקציה מתוך המפתח fn שבאובייקט obj שתחזיר לנו את הערך 5

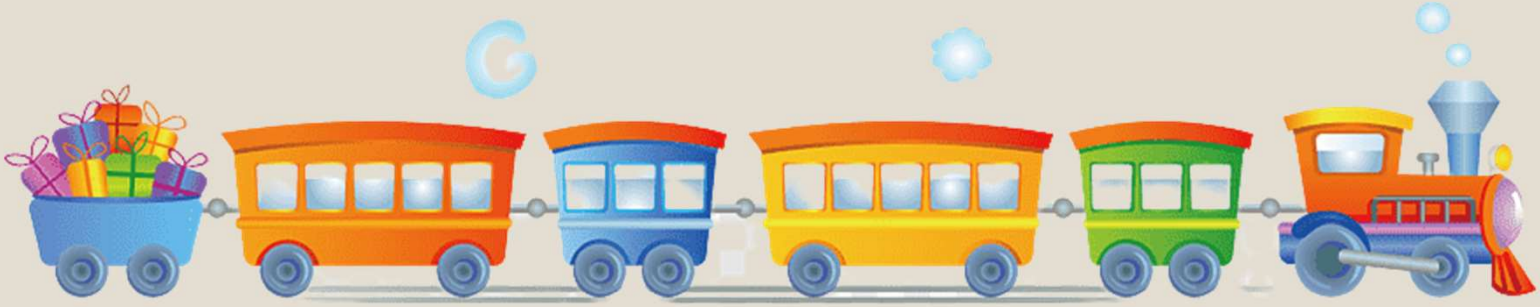
```
obj.fn()
```

איברים במערך

- צריך לחשוב על מערך כמו רכבת עם קרונות
- מס' האינדקס של המערך הוא המספר שבו מתחיל הקרון עד למספר בו מתחיל הקרון הבא
- האיברים במערך הם האנשים המאכלסים את הקרונות.
- זאת אומרת שאם אני בקרון הראשון אז הוא מתחיל מנקודה אפס עד אפס נקודה תשע תשע....

Const array = [“itemOne”, “itemTwo”]

- אם אני רוצה להתייחס לאיבר הראשון אז אני אפנה ל – array[0]



Object Destructor

- חילוץ מפתחות מתוך אובייקט והפיכתם לקבועים/ משתנים העומדים בפני עצמם

```
const obj = {
name: “David”,
Age: 207
}
```

הדרך הארוכה

```
const name = obj.name
const age = obj.age
```

הדרך הקצרה (object destructor)

```
const { name, age } = obj
```

By reference & by copy

ברגע שיצרנו אובייקט נשמר לו מקום בזיכרון והאובייקט שיצרנו מצביע על המקום הזה

כל הפניה לאובייקט היא בעצם הפנייה למקום בזיכרון כך ששינוי אחד מהאובייקטים ישנה את כל האובייקטים

לדוגמה:

```
const obj = {name: “David”}
const obj2 = obj
obj2.name = “Yossi”
```

הערך החדש ישתנה גם ב obj וגם ב – obj2 כך שהערך של המפתח name יהיה “Yossi”

אם ארצה להעתיק את הנתונים של האובייקט לתוך אובייקט חדש אני צריך לבצע shallow copy או deep copy

דוגמה:

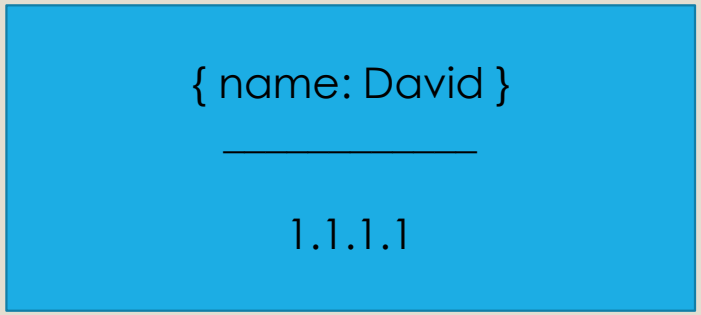
```
const obj3 = {…obj}
```

לאחר העתקת האובייקט בצורה הזאת נוצר מקום חדש בזיכרון לאובייקט המועתק ושינוי של פרמטרים בתוכו לא ישפיע על האובייקט המקורי.

Object by reference

Object By reference

- const obj = { }
- obj.name = “David”



- const second = obj
- second.age = 5

Object by copy

Object by copy

- const second = { …obj }
- second.age = 5”

{ name: David, age: 5 }

1.1.1.2

- const obj = { }
- obj.name = “David”

{ name: David }

1.1.1.1

Shallow & Deep copy

הקיצור דרך

```
const obj = {
  Key : {
          Name: “David”
          }
  }
```

הדרך הארוכה:

```
const address = { city: “tel-aviv”}

const user = {
name: “David”,
address: address
}
```

Object by copy

const user = {
name: “David”,
address: address
}

const address = { city: “tel-aviv” } ◦

{
name: David,
address: address
}

1.1.1.2

{ city: “tel-aviv”}

1.1.1.1

Deep copy