**Lesson 40-41 – Basic Node.js**

#### Lesson 40 - Node.js, Global, Naming, IIFE & Modules, NPM

# Node.js – סביבת הרצה של js מחוץ לדפדפן

* Ryan Dahl בנה תוכנה הנקראת "node.exe" (שאח"כ שונתה ל Node.js) שיודעת לתקשר עם מנוע V8 של כרום על ידי שפת js.

מקבל קוד js

מעביר אותו למנוע של דפדפן כרום

Node.js

V8 engine

* בזכות node.js אפשר להתשמש בשפת js בצד שרת.
* בנוסף בעוד שלדפדפן יש הגבלות על מחשב הלקוח (דפדפן לא יכול לגשת למסד נתונים או לשנות תיקיות על מחשב הלקוח) לתוכנת exe אין הגבלה בשליטה על המחשב

package.json - הגדרות של הפרויקט שלנו

dependencies – תוספים שהתקנו בפרויקט

יצירת קובץ node:

1. מתקינים את node דרך terminal
2. יוצרים קובץ pakage.json על ידי הפקודה npm init

# Global - האובייקט הראשי ב scope של node (מקביל ל window)

* דרך Global אפשר לקבל מידע על המחשב שלנו ((env, על Node (דרך process)
* כל הפונקציות נמצאות תחת אובייקט global

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Example 1 | Example 2 | Example 3 | Usage |
| ***Lowercase Camel*** | firstName | myBestFriend | age | משתנים ופונקציות |
| ***Uppercase Camel*** | FirstName | MyBestFriend | Age | שמות מחלקות |
| ***Snake*** | first\_name | my\_best\_friend | age | שמות משתנים ופונקציות בשפת PHP |
| ***Kebab*** | first-name | my-best-friend | age | שמות קבצים ושמות ישויות ב-CSS |

Naming Convention – מוסכמה לנתינת שמות למשתנים

# IIFE & Modules

* Node.js עוטפת באופן אוטומטי את הקוד ב IIFE.
* לכן כדי להשתמש בקוד בין דפים שונים אנחנו צריכים להחצין אותו על ידי יצירת modules

החצנה נעשית על ידי הפקודה:

module.exports

**הראינו שלוש דרכים להחצין קוד:**

1. החצנה של אוסף של פונקציות – ע"י יצירת אובייקט:

module.exports = {

  add, //  equel to add: add

  sub,

  mul,

  div

}

* העיקרון הוא שמחצינים רק אלמנט אחד.
* לכן אם רוצים להחצין כמה אלמנטים (שאינם תחת class ) דרך טובה היא להשתמש באובייקט

כדי להתשמש בקוד שהחצינו בדף בו אנחנו נשתמש בפקודה require:

const calc = require("./calc")

* הפקודה require מקבילה לפקודה Import מ react (לא משתמשים ב Import כיוון שnode עדיין לא תומכת בה)
* משתנה calc מקבל לתוכו את מה שאנחנו מחצינים מדף (./calc)
* במקרה הזה calc הוא אובייקט. לכן כדי לקרוא לפונקציות שהחצנו צריך להשתמש בסינטקס של אובייקט:

console.log(calc.add(10, 2))

console.log(calc.sub(10, 2))

console.log(calc.mul(10, 2))

console.log(calc.div(10, 2))

1. החצנה של פונקציה בודדת - אין צורך להכניס לאובייקט

module.exports = printArray;

* כדי השתמש בפונקציה נשתמש בפקודה require:

const printArray = require("./print-array")

* כיוון שהקוד המוחצן הוא פונקציה בודדת המשתנה אליו הכנסו את הקוד הופך לפונקציה:

printArray([3, 5])

1. החצנה של class - בדומה לפונקציה בודדת אין צורך להכניס את ה class לאובייקט

module.exports = Cat;

* המשתנה אליו מכניסים את הקוד הופך ל class ולכן כדי להשתמש ב אנחנו צריכים ליצור מופע –אובייקט חדש

const Cat = require("./cat")

const myCat = new Cat("Mitsi", 4)

myCat.display()

Build-in Modules – modules שאנחנו מקבלים במתנה מ node - לדוגמה os

const os = require("os");

* כיוון ש os הוא build-in היבוא שלנו נעשה ללא גישה לתיקייה ((./

npm Modules – ב npm ישנם הרבה modules שאפשר להוריד ולהשתמש

#### Lesson 41 - Async-await, File System

# asyinc & await

* צורת כתיבה שמבוססת על פונקציות שמחזירות Promise, העוזרת לנו לברוח ממצב של callback Hell – מצב שבו יש לנו שרשור של אירועים אה-סינכרוניים

**דוגמה בכיתה – שימוש ב module redline כדי להמחיש callback Hell:**

1. readline הוא build-in module

const readline = require("readline");

1. readline מצריך הגדרה ראשונית של ערכים createInterface() אותם נאתחל לתוך משתנה ui

const ui = readline.createInterface({

    input: process.stdin, // The Standard Input (מאיפה לקלוט מידע)

    output: process.stdout // The Standard Output (לאיפה להציג את המידע)

* השורות האלו מגדירות שאנחנו מקבלים ומציגים את המידע ב console.

1. על ידי readline אנחנו יכולים לדמות בצד שרת קבלת מידע מ user

* question() – פונקציה שמציגה שאלה ל user – מקבלת string (השאלה) ו callback – עיבוד של ה input שהמשתמש יכניס.

ui.question("Enter your first name: ", firstName => {

    console.log("Your first name is: " + firstName);

    ui.question("Enter your last name: ", lastName => {

        console.log("Your last name is: " + lastName);

    });

});

* אפשר לראות התחלה של שרשור callbacks – כל בקשה של מידע דורשת עד בקשה של question ועוד כתיבה של callback (ובמקרה הזה ביקשנו רק שני פרטים!)
* בסוף הפונקציה יש צורך לבצע סגירה – close() אחרת הפונקציה ממשיכה לרוץ
* ui.close();

חשוב לדעת - הפונקציות שאנחנו מפעילים עובדות בצורה סינכרונית ואה-סינכרונית:

* הפונקציה question ממתינה ל Input של ה User והקוד אחריה מחכה (סינכרוני).
* עם זאת, הקוד שמחוץ לפונקציה לא נתקע וממשיך לרוץ (אה-סינכרני)

1. כדי שיהיה יותר נוח לעבוד עם ui נעטוף אותה בתוך פונקציה (wrapper)

* כיוון שהפונקציות עובדות בצורה אה-סינכרונית אפשר לכתוב אותם בפורמט של promise (מה שמקל את הכתיבה של ה callback)

function getValue(messageToShow) {

    return new Promise((resolve, reject) => {

        try {

            ui.question(messageToShow, value => resolve(value));

        }

        catch (err) {

            reject(err);

        }

    });

}

function close() {

    ui.close();

}

1. נחצין את מה שכתבנו

module.exports = {

getValue,

close

};

1. בדף app נקרא למה שכתבנו לתוך משתנה (input)

const input = require("./input");

input.getValue("Enter your first name: ")

    .then(firstName => {

        console.log("Your first name: " + firstName);

        input.getValue("Enter your last name: ")

            .then(lastName => {

                console.log("Your last name: " + lastName);

                input.close();

            })

            .catch(err => console.log(err.message));

    })

    .catch(err => console.log(err.message));

* גם כאן על כל Input שאנחנו רוצים אנחנו צריכים לכתוב then נוסף!
* עדיין נשארנו במצב של שרשור callbacks

**הפיתרון – שימוש במנגנן async ו await.**

async - ממוקמת לפני הפונקציה

תפקיד - ממירה את שהפונקציה מחזירה (return) לאובייקט Promise

לדוגמה -

async function f() {

return 1;

}

f().then(alert); // 1

* כיוון ש f() מחזירה Promise אפשר לקבל את ה return לתוך then.
* עכשיו נוכל להתשמש בחלק השני await.

**await** - נכתבת לפני קריאה לפונקציה שמחזירה Promise

תפקיד – ל await יש 3 פעולות: 1. גורמת לקוד הפנימי של הפונקציה להמתין עד שהפקודה מתבצעת

1. היא משחררת את הזרימה של שאר התוכנית
2. מחלצת את מה שחוזר מ resolve.

פתרון בכיתה – הפונקציה getValue שכתבנו מחזירה promise

1. לפני getName נוסיף async

async function getNames() {

1. נעטוף את הקוד במנגנון try-catch כדי לתפוס קריסות

  try {

    const firstName = await input.getValue("Enter your first name: ");

    const lastName = await input.getValue("Enter your last name: ");

    console.log(`Your first name: ${firstName} Your lasr name: ${lastName}`);

  } catch (err) {

    console.log(err.message);

  } finally {

    input.close();

  }

}

1. נקרא הפעם ל getValue() תחת await –

* הקוד הפנימי בפונקציה ימתין שה user י כניס מידע
* מה שחוזר מ resolve נכנס לתוך משתנה.

1. את מה שקיבלנו אנחנו יכולים להציג ב console.