24.5.22

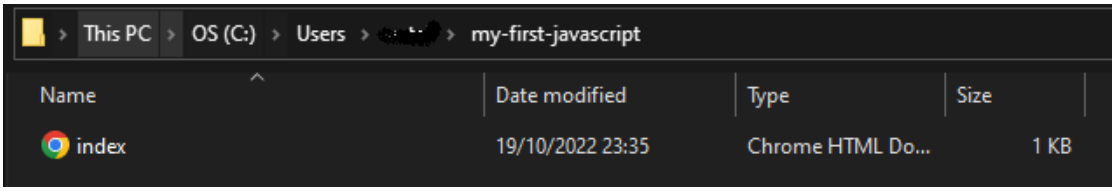
EASS שיעור 12- CSS ,Promises ,JavaScript

JavaScript:

- שפה סטנדרטית ל-UI.
- כדי שנוכל להשתמש בjavascript באתר, יש להטמיע את הקוד שלו בתוך קוד של HTML.
  ניתן לעשות זאת בעזרת תגית של HTML שנקראת <script>. דוגמה:

```
[redacted]:/mnt/c/Users/[redacted]$ mkdir my-first-javascript
[redacted]:/mnt/c/Users/[redacted]$ cd my-first-javascript
[redacted]:/mnt/c/Users/[redacted]/my-first-javascript$ vi index.html
```

```
<html>
   <head>
      <title>Hello World</title>
      <script>
         alert("Do not use alerts! It is annoying");
      </script>
   </head>
   <body>
      <h1>Below is rendered HTML using javascript</h1>
      <script>
         document.write("<h2>Hello world</h2>");
         document.write("<p>This document was last modified on "
             + document.lastModified + ".</p>");
      </script>
   </body>
</html>
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```



24.5.22

- בדומה לאיך שהשתמשנו ב-ipython כדי לכתוב ולקבל פלט בשפת python דרך WSL, נשתמש ב-node כדי לכתוב קוד ב-JavaScript.
תחילה נבצע את ההתקנה באמצעות פקודת sudo apt install nodejs או brew install node
ואז ניתן להשתמש בכלי node כך:

```
[redacted]:/mnt/c/Users/[redacted]/my-first-javascript$ node
> for (var i=1; i<=3; i++) { console.log(i); }
1
2
3
undefined
```

כדי לצאת מ-node נלחץ על ctrl+c במקלדת פעמיים או שנרשום exit. (נקודה ואז exit)

גם לולאת while פועלת בשפה זו כמו בשפת C או JAVA.

- השוואות בין משתנים בשפה עובדים בצורה הבאה:
אם יש == בין שני משתנים שאינם מאותו סוג, מתבצעת המרה (casting) של המשתנה השמאלי לסוג של המשתנה הימני
אם יש === בין שני המשתנים לא מתבצעת המרה של סוגי המשתנים.

```
> intNumber = 2
2
> charNumber = "2"
'2'
> intNumber == charNumber
true
> intNumber === charNumber
false
```

- נגדיר את המשתנים עם var שייקבע אוטומטית את סוג המשתנה בהתאם לערך שנכניס אליו:

```
var num1 = 5; // integer
var num2 = 3.14159 // floating point number
var phrase = 'Hello world!'; // string
var flag = true; // boolean value
```

- בדיקת סוג משתנה:

```
> typeof(charNumber)
'string'
> typeof(intNumber)
'number'
```

- לולאת foreach – עוברת על ערכים איטרביליים במבנה נתונים מסוג object (כמו רשימה, מערך וכו')

```
var cars = {car1:"Saab", car2:"Volvo", car3:"BMW"};

for (x in cars) {    //returns "car1", "car2", and "car3" as strings
  console.log(cars[x]);              //returns the value of each property
}
```

24.5.22

ההבדל בין of ל-in בלולאה מסוג זה:

```
> a = [10,20,30,40,50]
[ 10, 20, 30, 40, 50 ]
> for (i of a) {print(i);}
10
20
30
40
50
undefined
> for (i in a) {print(i);}
0
1
2
3
4
undefined
```

- Dictionary בשפה זו הוא JSON.

```
var obj = {name: 'John', age: 30};
console.log(obj); // {name: 'John', age: 30}
```

- ניתן להגדיר מתודה כערך:

```
var car = {
    type: 'sedan',
    color: 'blue',
    drive: function() {
        method: console.log('The car is driving');
    }
};
```

24.5.22

- מחלקות ובנאים:

```
class Person {

  name;

  constructor(name) {
    this.name = name;
  }

  introduceSelf() {
    console.log(`Hi! I'm ${this.name}`);
  }

}


const john = new Person('John');

john.introduceSelf(); // Hi! I'm John
```

- ירושה:

```
class Professor extends Person {

  teaches;

  constructor(name, teaches) {
    super(name);
    this.teaches = teaches;
  }

  introduceSelf() {
    console.log(`My name is ${this.name}, and I will be your ${this.teaches} professor.`);
  }

  grade(paper) {
    const grade = Math.floor(Math.random() * (5 - 1) + 1);
    console.log(grade);
  }

}
```

24.5.22

:Promises

אובייקט ש"מבטיח" להחזיר ערך בשלב מסויים.
בנוי כקטע קוד שחייב להמתין לתוצאה של קטע קוד אסינכרוני (שיכול לקחת זמן).

ל-Promise יש שלושה מצבים:

1. Pending – מצב התחלתי, הפעולה לא הסתיימה.
2. Fulfilled – הפעולה הסתיימה וה-Promise החזיר ערך
3. Rejected – הפעולה נכשלה וה-Promise מחזיר את סיבת הכשלון.

()Promise.then – מקבלת שני ארגומנטים- אחד למקרה שהpromise יצליח ואחד למקרה שהpromise ייכשל.

```
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

כאשר ניצור את אובייקט promise ונגדיר בו מהי הפעולה שיכולה לקחת זמן, ובהתאם לתוצאה שלה ()Promise.then ייקבע אם נכשל או הצליח.

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});
```

מקור: https://www.w3schools.com/js/js_promise.asp

דוגמה נוספת:

```
2 //curl 'http://httpbin.org/get'
3 prom = fetch("http://httpbin.org/get");
4 // prom.state == "pending"
5 console.log("instant") // prom is still pending
6
7 prom.then(response => response.json())
8     .then(data => console.log(data));
```

בשורה 3 הגדרנו את prom כבקשה שנשלחה לשרת. הזמן שייקח עד שתתקבל תשובה מהשרת אינו ידוע.
לבנתיים, הפעולה בשורה 5 מתרחשת, למרות שהסטטוס של prom הוא עדיין pending , כי היא אינה תלויה בתשובה מהשרת.
לעומתה, הפעולה בשורות 7 ו-8 ממתינה עד שתחזור תשובה ושהסטטוס של prom ישתנה ל-Fulfilled או Rejected , כי פעולה זו היא ()Promise.then וכל שאר הקוד שאחריה גם ממתין עד שהיא תתממש.

הערה- ניתן לשרשר כמה then אחד אחרי השני שכל אחד ממתין לסיום פעולתו של הקודם.

24.5.22

:CSS

שפה שמעצבת ומגדירה מאפיינים עבור תגיות של HTML.

לדוגמה:

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

כל מה שמופיע בתוך תגית body יהיה עם רקע כחול בהיר, כל הטקסט שמופיע בתגית h1 יהיה לבן וממורכז, כל הטקסט שיופיע בתגית p יהיה בפונט verdana ובגודל 20 פיקסלים.

ניתן "לדרוס" או להוסיף בסעיף נפרד שינויים עבור אותה תגית. לדוגמה:

```
p {
  color: red;
}

body {
  background: #ffffff;
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
}

a {
  color: #0088cc;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

p {
  font-size: 13px; line-height: 1.6em; margin-bottom: 10px;
}
```

כאן נוסף לסעיף של תגית p שינוי נוסף בגודל הפונט.

24.5.22

פונקציות אסינכרוניות ב-JavaScript:

```
> async function addAsync(x, y, callback) { var result = x + y; await new Promise(r => setTime
out(r, 2000)); callback(result);}
```

נגדיר פונקציה שמקבלת שני משתנים x,y ונגדיר שהיא מחזירה callback. הפונקציה מבצעת פעולת חיבור בין שני המשתנים, ממתינה שתי שניות וה-callback שלה הוא ערך משתנה תוצאת החיבור result.

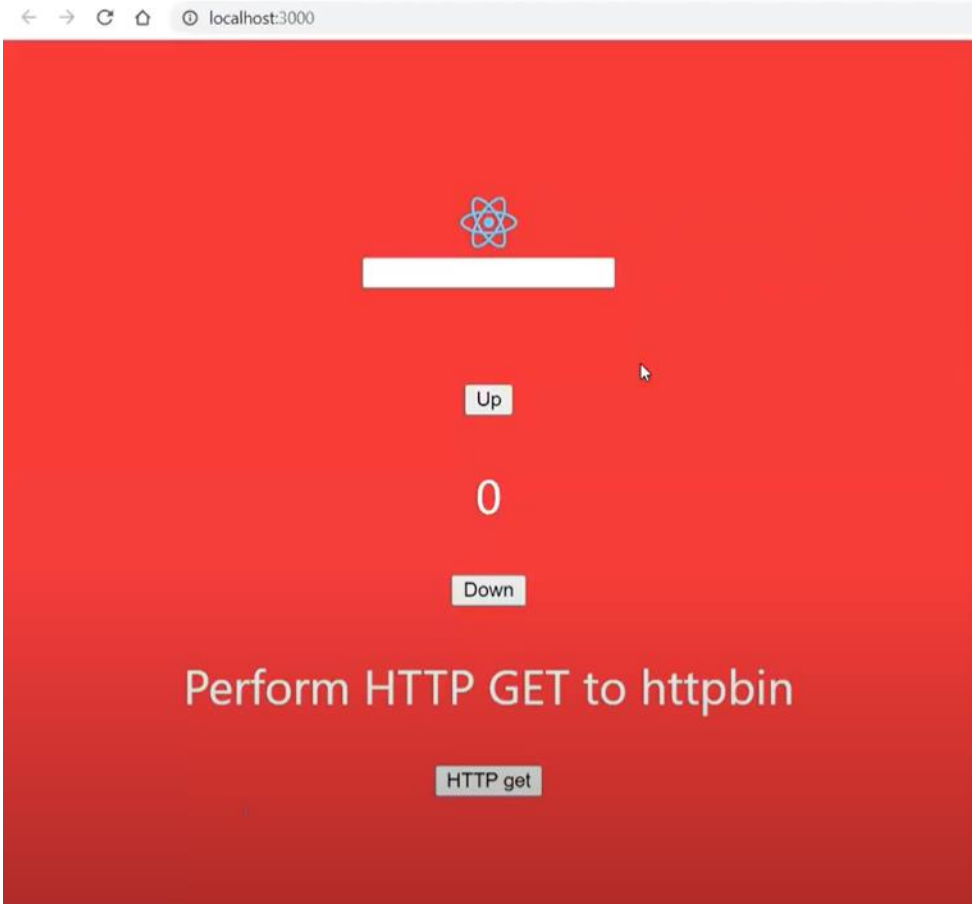
```
> addAsync(1,2,function(result){console.log(result);})
Promise {
  <pending>,
  [Symbol(async_id_symbol)]: 10815,
  [Symbol(trigger_async_id_symbol)]: 5,
  [Symbol(destroyed)]: { destroyed: false }
}
> 2+2
4
> 3
```

מכיוון שהפונקציה היא אסינכרונית, בזמן שהיא רצה אנו יכולים לבצע עוד פעולות (כמו חיבור של 2+2) עד שתחזור התשובה שלה.

24.5.22

ReactJS:

ספרייה ב-JavaScript שמשמשת לכתיבת UI.



(הקוד המלא עבור דף זה נמצא במצגת שיעור בגיטהאב https://github.com/EASS-HIT-2022/Part-A/blob/main/lectures/slides_part_a.md)

נזמן את הספרייה בקובץ JavaScript:

import React, { Component } from "react";

```
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} alt="Logo" width="10%" />
          <input type="text" onChange={this.handleChange} value={this.state.inputText} />
          <p>{this.state.inputText}</p>
          <button name="Up" onClick={this.handleClick}>
            Up
          </button>
          <p>{this.state.counter}</p>
          <button name="Down" onClick={this.handleClick}>
            Down
          </button>
          <p>Perform HTTP GET to httpbin</p>
          <button onClick={this.handleGet}>HTTP get</button>
        </header>
      </div>
    );
  }
}
```

24.5.22

בחלק של render נכתוב את כל התגיות HTML שמייצרות את הדף, בתוך הסוגרים המסולסלים שיש בתגיות נכתוב קוד JavaScript, שבמקרה שלנו הוא מתודות שנקרא להן כשמתבצעת פעולה (כשלוחצים על כפתור, כשמכניסים טקסט לתיבת טקסט).

מעל לחלק של render , בתוך מחלקה שיורשת מ-Component של React, נכתוב את כל המתודות שלהן קוראים. לדוגמה המתודה שמתרחשת כשלוחצים על כפתור Up או כפתור Down:

```
handleClick = (event) => {
  console.log(event.target);
  const val = event.target.name === "Up" ? 1 : -1;
  this.setState({
    counter: this.state.counter + val,
  });
};
```

ה-event הוא אובייקט. הוא מכיל בתוכו בין היתר את target שהוא אובייקט ה-HTML שקרא למתודה (במקרה זה הכפתורים).