

More topics in JavaScript

Closure

Callback function

Promise

Fetch

Closure

צד ל��וח - יעל סלע זעירא

Global Variables

- לפונקציה יש גישה לכל המשתנים שלה
- כמו בדוגמה הבאה : המשתנה a

```
<body>
  <p id="demo"></p>

  <script>
    myFunction();

    function myFunction() {
      let a = 4;
      document.getElementById("demo").innerHTML = a * a;
    }
  </script>
</body>
```

Global Variables

- לפונקציה יש גישה לכל המשתנים המוגדרים מחוץ לפונקציה
- כמו בדוגמה הבאה : המשתנה a

```
<body>
  <p id="demo"></p>
  <script>
    let a = 4;
    myFunction();

    function myFunction() {
      document.getElementById("demo").innerHTML = a * a;
    }
  </script>
</body>
```

Global Variables

- משתנים המוגדרים לא `var, let, const` יהיו גלובליים

```
<body>
  <p id="demo"></p>

  <script>
    myFunction();
    document.getElementById("demo").innerHTML = a * a;

    function myFunction() {
      a = 4;
    }
  </script>
</body>
```

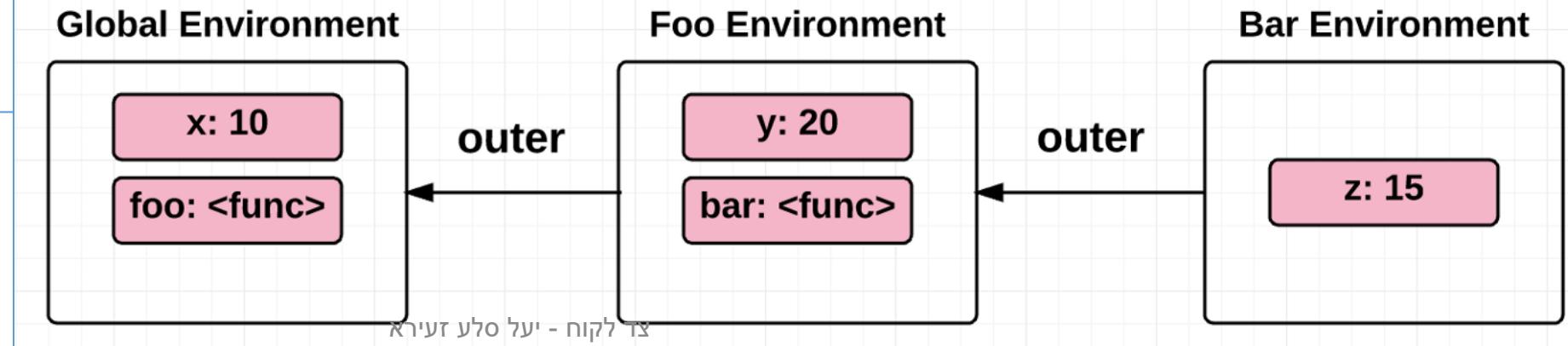
Closure

- Closure היא פונקציה פנימית (inner function) אשר יש לה מצביע למחצית שוטף אותה (outer function)
- במילים אחרות clouser נוטן גישה ל:
 - ✓ למשתנים של הפונקציה עצמה
 - ✓ למשתנים של הפונקציה שעוטפה אותה
 - ✓ למשתנים הגלובליים
- פונקציה כזו היא פונקציה ש"זכרת" את הסביבה שיצרה אותה

Scope

מה ידפו?

```
<script>  
var x = 10;  
  
function foo() {  
    let y = 20;  
    function bar() {  
        let z = 15;  
        return x + y + z;  
    }  
    return bar;  
}  
let a = foo();  
console.log(a())  
</script>
```



מה יודפס?
איזה x ילכוד במקרה זה ?

```
var x = 10;  
  
function foo() {  
    var y = x + 5;  
    return y;  
}  
  
function bar() {  
    var x = 2;  
    return foo();  
}  
console.log(foo());  
console.log(bar());
```

```
<script>
  function sayHello() {
    let say = function () { console.log(hello); }
    // Local variable that ends up within the closure
    let hello = 'Hello, world!';
    return say;
  }
  let sayHelloClosure = sayHello();
  sayHelloClosure(); // 'Hello, world!'
</script>
```

משתנה hello מוגדר אחרי פונקציה אונומית אך לפונקציה עדין יש גישה אליו, זאת מכיוון שהמשתנה כבר הוגדר כמשתנה הנמצא ב scope של הפונקציה בזמן שהפונקציה נוצרה ולכן כבר זמין בזמן הריצעה של sayHello.

אנו רואים שיש לנו גישה למשתנים של הפונקציה העוטפת גם לאחר שהפונקציה חזרה.

```
var x = 10;
function foo(a) {
var b = 20;

function bar(c) {
var d = 30;
return boop(x + a + b + c + d);
}

function boop(e) {
return e * -1;
}

return bar;
}

var moar = foo(5); // Closure
moar(15);
```

```
var x = 10;
function foo(a) {
var b = 20;

function bar(c) {
var d = 30;
return boop(x + a + b + c + d);
}

function boop(e) {
return e * -1;
}

return bar;
}

var moar = foo(5); // Closure
moar(15);
```

הפונקציה `foo(5)` ממחזירה את `bar` לתוך המשנה `moar` קרייה ל `(15) moar` מפעילה את `bar` עם הפרמטר `15` , בתווך `e` קיימ כבר הערך `5` מהקרייה ל `foo(5)`.

`bar` מפעילה את `boop` והערך המוחזר הוא:
$$(10+5+20+15+30)*-1$$

Use Cases of Closures

Data Encapsulation

- Closure מאפשר **ניהול המידע** – על ידי הסתרה של משתנים מסוימים, אפשרות שינוי משתנים רק במקום מסוים
- דוגמא:

```
function createCounter() {  
    let count = 0;  
  
    function increment() {  
        count++;  
        return count;  
    }  
  
    return increment;  
}
```

```
const counter1 = createCounter();  
const counter2 = createCounter();  
  
console.log(counter1()); // Output: 1  
console.log(counter2()); // Output: 1  
  
console.log(counter1()); // Output: 2  
console.log(counter2()); // Output: 2
```

מתבצעת אנקופולציה למשתנה ה count

הגישה היחידה למשתנה היא על יד

שיבצע הגדולה ויחזיר את הערך.

קריאה ל createCounter ממחזירה את . increment

מספר קריאות ל createCounter יחזרו closures
שמות שונים לכל אחד state שלו.

Use Cases of Closures

```
<button class="counter">A</button>
<button class="counter">B</button>

function createCounter(name) {
  let count = 0;

  return function () {
    count++;
    console.log(` ${name}: ${count}`);
  };
}

document.querySelectorAll(".counter").forEach(btn => {
  btn.addEventListener("click", createCounter(btn.innerText));
});
```

Event handlers with preserved state

Without closure:

```
let count = 0;
button.addEventListener("click", () => {
  count++;
});
```

Button A remembers its own count

Button B remembers its own count

Each handler has its own closure

Use Cases of Closures

Functional Programming

אפשר ייצרת פונקציות שמקבלות פונקציות כפרמטרים ומחזירות פונקציות Closure

```
function createMultiplier(factor) {  
    return function (x) {  
        return x * factor;  
    };  
}  
  
const double = createMultiplier(2);  
const triple = createMultiplier(3);  
  
console.log(double(5)); // Output: 10  
console.log(triple(5)); // Output: 15
```

קוד צפוי ויציב יותר

פחות באגים

בדיקות פשוטות

תחזוקה קלה

Callback function

Callback function

פונקציה אשר נשלחת כפרמטר לפונקציה אחרת, וזו נקראת בתוך הפונקציה אליה הגיעו כפרמטר.

```
function operation(val1, val2, callback) {  
    callback(val1, val2)  
}  
  
function sum(a, b) {  
    console.log(a + b)  
}  
  
function divide(a, b) {  
    console.log(a / b)  
}  
  
operation(6, 5, sum)  
operation(9, 3, divide)
```

CallBack Function Use cases- Asynchronous Operations

יצירת אירוע בצורה אסינכרונית

```
<script>  
  
setTimeout(sayHi, 3000);  
  
function sayHi() {  
    console.log("Hi")  
}  
</script>
```

```
<script>

setTimeout(function () { myFunction("I love You !!!"); }, 3000);

function myFunction(value) {
    document.getElementById("demo").innerHTML = value;
}

</script>
```

CallBack Function Use cases

קריאה לפונקציה כתגובה לאירוע

```
document.getElementById("myButton").addEventListener("click", function ()  
{  
    console.log("Button clicked.");  
});
```

-
- נראה מיד שימוש בcallback עבור תגובה לקריאה http

Fetch API (Promise)

Fetch API

פונקציה המאפשרת לבצע קריאות HTTP ולקבל תגובה

API

Application Programming Interface

סט של כללים וכליים המאפשרים לתוכנות שונות לדבר זו עם זו ולחולוק מידע
גם אם הן מפותחות על ידי ארגונים שונים וגם אם הן רצות על פלטפורמות
שונות.

Web API Basics

- URL שהAPI חושף, כל endpoint מיצג פונקציה שהAPI יבצע
- XML , JSON - Request and Response Format
- הרבה פעמים יש צורך שרק משתמשים מורשים – Authentication and Authorization
- יכולו לגשת לAPI. לצורך כך משתמשים ב Token או API Key

Fetch API

- javascript מאפשר לנו לבצע קריאות HTTP ב Fetch api
- כל הבראוזרים הפופולריים כיום תומכים ב fetch api
- Fetch api מספק פונקציה גלובלית שניתן להשתמש בה הנקראת `fetch`, והיא דואגת להחזיר `resources` מה server

Syntax

```
fetch(URL [, options])
```

URL : a URL object that represents the path **of** the resource to be fetched

Options(optional): Any further options such as:

- Method: The request method is either GET POST PUT DELETE.
- Headers : for example '**Content-Type**': 'application/json'
- Body: json string
- Mode
- Credentials
- Cache

How fetch() works

```
<script>
fetch('url')
  .then(response => {
    //handle response
    return response.json()
  })
  .then(data => {
    //handle data
  })
  .catch(error => {
    //handle error
  });
</script>
```

Calling an External API Using Fetch

- **נשתמש ב API חינמי הנקרא JSONPlaceholder API**
- **אינו דורש אימות משתמש או API key**
- <https://jsonplaceholder.typicode.com/guide/>

Fetch

- נשתמש בפונקציה fetch של javascript על מנת לפנות ל אPI חיצוני שיחזיר נתונים,
- את הנתונים נדפיס לוג

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    // Check if the response is successful
    return response.json(); // Parse the JSON from the response
  })
  .then(data => {
    // Use the data from the response
    console.log(data);
  })
  .catch(error => {
    // Handle any errors
    console.error('Error:', error);
  });
};
```

Fetch with error handling

```
// Fetch data from the JSONPlaceholder API
fetch('https://jsonplaceholder.typicode.com/posts')
.then(response => {
  if (!response.ok) {

    throw new Error(`HTTP error! Status: ${response.status}`);
  }
  // Return the response data as JSON
  return response.json();
})
.then(data => {
  // Handle the JSON data
  console.log('Fetched Data:', data);
  // Example: Display the titles of the posts
  data.forEach(post => {
    console.log(`Post ${post.id}: ${post.title}`);
  });
})
.catch(error => {
  console.error('There was a problem with the fetch operation:', error);
});
```

Getting a resource

```
fetch('https://jsonplaceholder.typicode.com/posts/1')

.then((response) => response.json())

.then((json) => console.log(json))

.catch(error => {

  console.error('There was a problem with the fetch operation:', error);

});
```

Getting resources

```
fetch('https://jsonplaceholder.typicode.com/posts')

.then((response) => response.json())

.then((json) => console.log(json))

.catch(error => {

  console.error('There was a problem with the fetch operation:', error);

});
```

```
<body>
<h1>Post Titles</h1>
<ul id="post-list"></ul>
<script>
// JavaScript code goes here
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return response.json();
})
.then(data => {
  console.log(data);
  const postList = document.getElementById('post-list');
  data.forEach(post => {
    const listItem = document.createElement('li');
    listItem.innerText = `Post ${post.id}: ${post.title}`;
    postList.appendChild(listItem);
  });
})
.catch(error => {
  console.error('Fetch error:', error);
});
</script>
</body>
```

תרגיל ביתה

ה זם אליו תפנו יחזיר לכם את רשימת החגים בארה"ב
אתם מעוניינים אך ורק בשמות של name ו date |
הציגו את המידע ל client ובדף html
להלן api endpoint

<https://date.nager.at/api/v3/publicholidays/2026/US>

שאלות?

PROMISE

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())
  .then(json => console.log(json))
  .catch(error => console.log(error));
```

ASYNC/AWAIT

```
async function runProcess() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');
    const json = await response.json();
    console.log(json);
  } catch (error) {
    console.log(error);
  }
}

runProcess();
```

צד לקוח - יעל סלע זעירא;

הצורה המועדףת על
המתכנתים היא יותר זורמת
עם הקוד