JS Functions

JAVASCRIPT

מה נלמד?

- מהי פונקציה?
- Function Declarations
- scope
- ternary operator
- רקורסיה
- Function expression
- Self-Invoking Functions
- closure

פונקציה ומתודה

פונקציה היא סט פקודות מאורגן במבנה אחד

- למשל פונקציה שבעת לחיצה על כפתור תפעיל alert שמציג את הערך בתיבת ה-input

מתודה היא סט פקודות מאורגן במבנה אחד, המשויך לאובייקט כלשהו

- למשל המתודה getElementById המשויכת ל-document

Function Declarations

```
function print() {
  console.log("Hello!");
}

print();
```

- פונקציה שלא מקבלת ולא מחזירה ערך

```
function calcSum() {
  let numbers = [10, 20, 30, 40];
  let sum = 0;
  for (let i = 0; i < numbers.length; i++) {
    sum += numbers[i];
  }
  return sum;
}
console.log(calcSum());
```

- פונקציה שלא מקבלת ערך אבל מחזירה ערך

Function Declarations

- פונקציה שמקבלת ערך אבל לא מחזירה ערך

```
var numbers = [10, 20, 30, 40];

function printArr(numbersArr) {
  console.log(numbersArr);
}

console.log(numbers);
```

- פונקציה שמקבלת ערך ומחזירה ערך

```
var numbers = [10, 20, 30, 40];

function printArr(numbersArr) {
  let sum = 0;
  for (let i = 0; i < numbersArr.length; i++) {
    sum += numbersArr[i];
  }
  return sum;
}

console.log(printArr(numbers));
```

scope

- משתנים לוקאליים מול משתנים גלובאליים

גלובאלי

לוקאלי

לוקאלי

```
var numbers = [10, 20, 30, 40];

function printArr(numbersArr) {
  let sum = 0;
  for (let i = 0; i < numbersArr.length; i++) {
    sum += numbersArr[i];
  }
  return sum;
}

console.log(printArr(numbers));
```

ternary operator

- משפט תנאי מקוצר בפונקציה

```
function checkNum(num) {
  return num > 10 ? "Bigger" : "Smaller";
}

console.log(checkNum(30));
console.log(checkNum(5));
```

```
Bigger
Smaller
```

פונקציה שנשלחת לפונקציה

- מה יודפס לקונסול?

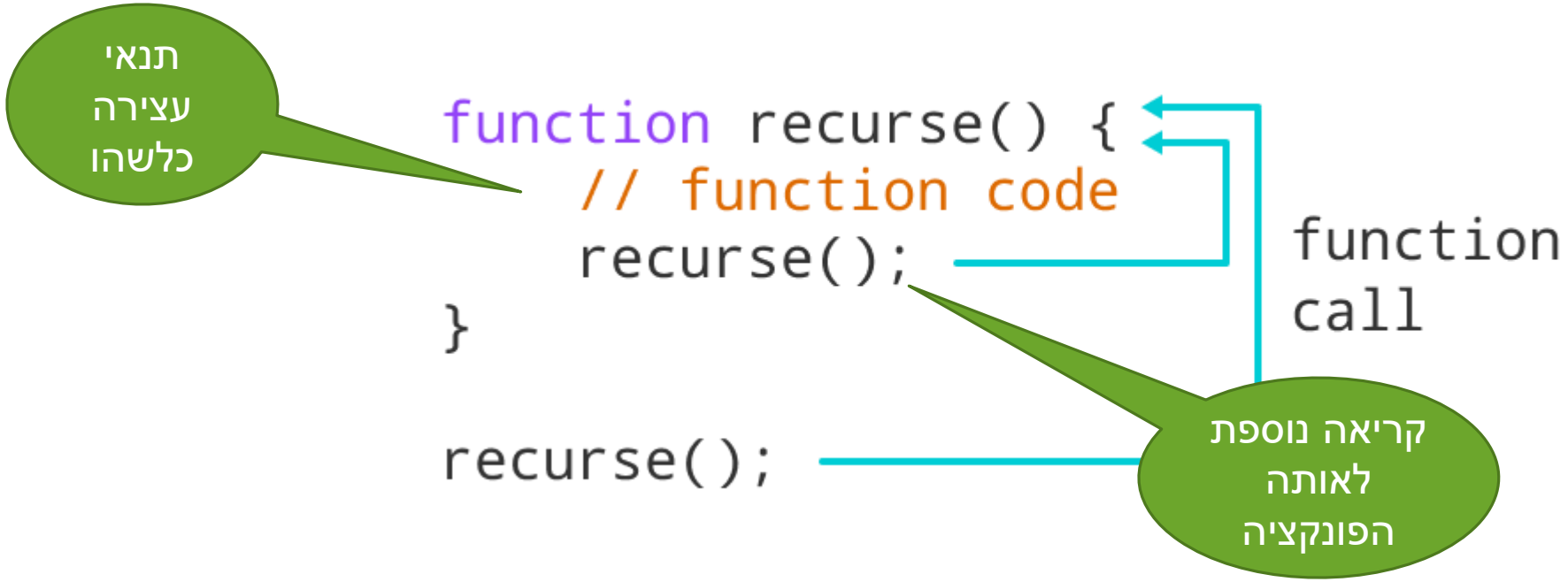
```
function multiple(x) {
  return x * 2;
}

function calc5(x, y, _multiple) {
  var num1 = (x % 2) + 1;
  var num2 = y % 5;
  return _multiple(num1 + num2);
}

var result = calc5(9, 7, multiple);
console.log(result);
```

רקורסיה

- כתבו רקורסיה בגוגל ☺
- רקורסיה היא אחת הטכניקות הנפוצות בעבודה מול מידע בכמות לא ידועה.
- פונקציה שקוראת ומפעילה את עצמה מבפנים בצורה מעגלית עד לנקודה בה היא מפסיקה כיוון ותנאי מסוים לא התקיים

רקורסיה

- מבנה הרקורסיה:



רקורסיה

- דוגמה 1: פונקציה שמחשבת סכום את כל המספרים מתחת למספר כלשהו.

```
function sumUpTo(x) {
  if (x == 0) return 0;
  return x + sumUpTo(x - 1);
}

console.log(sumUpTo(5));
```

15

רקורסיה

- דוגמה 2: פונקציה שמחשבת עצרת מתמטית (מכפלת כל המספרים מתחת למספר כלשהו)

```
function factorial(x) {
  if (x == 1) return 1;
  return x * factorial(x - 1);
}

console.log(factorial(6));
```

720

רקורסיה

- מה מבצעת הפונקציה הבאה? מה יודפס לקונסול?

```
function countDown(fromNumber) {
  console.log(fromNumber);
  countDown(fromNumber - 1);
}
countDown(3);
```

רקורסיה

- מה מבצעת הפונקציה הבאה? מה יודפס לקונסול?

```
function secret(num) {
  if (num == 0) {
    return 0;
  }
  return (num % 10) + secret(Math.floor(num / 10));
}
console.log(secret(162));
```

Function expression

- צורת הצהרה וכתיבה שונה של פונקציה
- פונקציה מהסוג הזה, יכולה להתאחסן בתוך משתנה
- פונקציה אנונימית

```
var fn = function () {
  console.log("Hello from function expression!");
};

fn();
```

```
Hello from function expression!
```

Function expression

- מה יודפס לקונסול?

```
fn();

var fn = function () {
  console.log("Hello from function expression!");
};
```

```
Uncaught TypeError: fn is not a function
    at index.html:91:7
```

בשונה מפונקציות רגילות, חוקי ה-hoisting של פונקציות לא חלים כאן

פונקציה שמחזירה פונקציה

- ניתן שפונקציה תחזיר פונקציה שמבצעת משהו
- יש משמעות לסדר שליחת הארגומנטים

```
function myFunc(str1) {
  return function (str2) {
    return str1 + str2;
  };
}

console.log(myFunc("My name is ")("Mor"));
```

```
My name is Mor
```

Self-Invoking Functions

- פונקציות (מהסוגים השונים) יכולות לקרוא לעצמן ולהוציא את עצמן לביצוע מידי
- הפונקציה רצה פעם אחת
- מתאים למקרים בהם נרצה לבצע קוד אתחול ראשוני מבלי להשתמש במשתנים ב-scope גלובלי

```
(function () {
  let message = "Hello";
  console.log(message + " you!");
})();
```

```
Hello you!
```

closure - סגירות

- אם נרצה שמשתנה מסוים יהיה שייך אך ורק לפונקציה מסוימת והקוד הגלובאלי לא יוכל לגשת אליו או לדרוס אותו, נהוג להשתמש ב-closure

Self-invoked function.
במשתנה add מאוחסנת
הפונקציה הפנימית

```
let add = (function () {
  var counter = 0;
  return function () {
    counter += 1;
    return counter;
  };
})();
// counter = 0
add(); // counter = 1
add(); // counter = 2
console.log(add()); // counter = 3
```

משתנה מקומי
בפונקציה החיצונית
שגלובאלי עבור
הפונקציה הפנימית.
מאותחל פעם אחת
בלבד

קריאות נוספות
לפונקציה
הפנימית