ECMA6

ECMAScript היא ספסיפיקציה לשפת סקריפט.

ECMA - ראשי תיבות של European Computer Manufacturers Association – התאחדות אירופאית ליצרני מחשבים. ארגון ללא מטרות רווח שמטרתו הנוכחית היא להתוות סטנדרטים לתקשורת. הסטנדרט המוכר ביותר של הארגון הוא ECMAScript. שפת JavaScript בנויה על בסיס הסטנדרטים של ECMAScript – כלומר ECMAScript הן ההוראות שלפיו JavaScript צריכה להתיישר.

בדיוק כמו גרסאות של כל שפות הפיתוח, גם ECMAScript יוצאת כל פעם בתקן חדש של תוספות וחידושים. תקן ECMAScript 5 פורסם בשנת 2009. בתקן הזה כבר ישנה תמיכה בפורמט JSON (בעבר היה נדרש להכניס פונקציות ייחודיות שיטפלו בפורמט הנתונים הזה). התקן האחרון של ECMAScript הוא תקן 6 אשר יצא לאור ביוני 2015.

תקן הזה כולל רפורמה משמעותית מאוד של השפה: שינויים רבים ותוספות פונקציונליות.

ניתן חלק את הרפורמה לשלוש קטגוריות עיקריות:

תיקון באגים ובעיות ב-ECMAScript 5

תוספות משמעותיות לשפה כדוגמת לולאות מיוחדות.

Syntactic Sugar – שינויים שנועדו להפוך את השפה ליותר קריאה.

תמיכת דפדפנים

לא כל הדפדפנים הטמיעו עדיין את ECMAScript 6, או יותר נכון, הם לא מריצים את JavaScript לפי התקן החדש. ככל שעובר הזמן והדפדפנים מתעדכנים, כך התמיכה שלהם בתקן הולכת וגדלה. נכון לכתיבת שורות אלו, לא כל הדפדפנים תומכים באותה המידה בתקן.

ישנם אתרים יעודיים להשוואת התמיכה בין הדפדפנים השונים.

החומר בקורס זה כולל נושאים שאינם קשורים ישירות ל ECMA6, אך הם חלק מהנושאים המתקדמים שיש להכיר ב JavaScript.

תכנות מונחה עצמים

אובייקטים

אובייקט הינו טיפוס נתונים מורכב המכיל ערכי נתונים רבים בתוך יחידה אחת, לעומת טיפוס פרימיטיבי בעל ערך של נתון אחד : מספר, מחרוזת, ערך בוליאני.

אובייקטים הם למעשה אוסף של תכונות, כאשר לכל תכונה יש שם וערך.

יצירת אובייקט

יצירת אובייקט נעשית ע"י האופרטור new ואחריו שם של פונקציית ctor אשר מאתחלת את האובייקט.

יצירת אובייקט ע"י new:

ניתן ליצור אובייקט חדש ע"י שימוש במילת המפתח new. לאחר מילת המפתח new יש לשים שם פונקציה אשר מגדירה את האוביקט החדש.

דוגמא:

```
var o = new Object();
```

יצירת אובייקט ע"י function:

ניתן ליצור אובייקטים באמצעות מילת המפתח function. באופן זה נוצר Constractore לאובייקט.

דוגמא:

```
function myFunction()
    {
    }

  myObject = new myFunction();
```

יצירת אובייקט ע"י literal object:

object literal מאפשר לשבץ את תיאור האובייקט באופן מילולי בקוד ה JS - בדרך דומה לזו שמשבצים נתון טקסטואלי בקוד, כמחרוזת.

object literal מורכב מרשימת תכונות, מופרדות ע"י פסיקים בתוך סוגריים מסולסלים. כל תכונה מורכבת משם , נקודותיים וערך.

למעשה, JSON הוא אובייקט literal, וכיום זהו הפורמט הפופולרי להגדרת אובייקט, בשל היותו חסכוני מאוד בתעבורת רשת.

דוגמא:

```
var circle = { x:0, y:0, radius:2 }
var person =
   {
     name: "sara cohen",
     age: 15,
     email: homer@simpson.com
```

```
    };
```

גישה לתכונות האובייקט

גישה לערך של תכונה של אובייקט נעשית ע"י אופרטור נקודה (.).
הערך מצד שמאל הינו שם האובייקט (מצביע לאובייקט).

הערך מצד ימין של הנקודה צריך להיות השם של התכונה, שחייב להיות מזהה, לא מחרוזת או ביטוי.
דוגמא (שימוש ב new):

```
var book = new Object();
```

קביעת תכונה של האובייקט

```
book.title = "JavaScript: The Definitive Guide";
```

דוגמא לתכונות נוספות:

```
book.chapter1 = new Object();
book.chapter1.title = "Introduction" ;
book.chapter1.pages = 25;
book.chapeter2 = {title: "Lexical Structure", pages:10};
```

קריאת ערכי תכונות מהאובייקט:

```
alert("Outline: "+ book.title + "\n\t" +
      "Chapter 1 " + book.chapter1.title + "\n\t" +
      "Chapter 2 " + book.chapter2.title);
```

מדוגמא זו, ניתן לראות כי יצירת תכונה חדשה של אובייקט יכולה להיעשות ע"י השמת ערך.
כמו כן ניתן לשנות ערך של תכונה ע"י השמה חדשה.

תכונה לא מוגדרת של אובייקט

במידה ונעשה ניסיון לקרוא ערך של תכונה שלא קימת יתקבל ערך JS מיוחד: **undefined**.

הסרת תכונה של אובייקט

```
delete book.chapter2;
```

יש לשים לב שהצבה של הערך undefined אינה זהה למחיקה, התכונה עדין קיימת, היא רק מכילה את הערך undefined .

Constractor

ה ctor נועד לאתחל את ערכי האובייקט, כאשר מגדירים באמצעות function.

פונקציית ה ctor ב JS-היא פונקציה עם שני מאפיינים מיוחדים:

1. הפונקציה נקראת ע"י האופרטור new.

2. הפונקציה מקבלת מצביע לאובייקט חדש ריק ע"י מילת המפתח this ואחראית לאתחול נכון של האובייקט החדש.

דוגמא:

```
function Rectangle(w,h)
{
   this.width = w;
   this.height = h;
}
```

קריאה לפונקציית ה ctor ליצירת שני אובייקטים מסוג מלבן:

```
var rect1 = new Rectangle(2,4);
var rect2 = new Rectangle(8.5,11);
```

יש לשים לב שפונקציית ה ctor משתמשת בארגומנטים, על מנת לאתחל את תכונות האובייקט. הוגדרה מחלקה של אובייקטים ע"י פונקציית ה ctor המתאימה, כל האובייקטים נוצרים ע"י ה ctor Rectangle() עובדה המבטיחה שיתבצע אתחול התכונות width ו height.

בשל העובדה כי פונקצית ה ctor מגדירה מחלקה של אובייקטים, חשוב לתת ל ctor שם המתאר את מחלקת האובייקטים שהיא מייצרת.

פונקצית ה ctor בדר"כ אינה מחזירה ערך, ואם כן, האובייקט המוחזר מהווה את ערכו של ביטוי ה new-במקרה כזה האובייקט בעל הערך this נמחק.

מתודות

מתודה היא פונקציה של JS הנקראת דרך אובייקט.

אם קימת פונקציה f ואובייקט Obj ניתן להגדיר מתודה בשם myMethod כך:

```
 obj.myMethod = f
```

בעת הפעלת המתודה, תופעל הפונקציה f.

הערה: אין להוסיף סוגריים בשם הפונקציה.

ניתן גם לכתוב ישירות את הפונקציה במתודה:

```
obj.myMethod = function(param1, param2)
{
return param1 * param2;
}
```

קריאה למתודה:

```
obj. myMethod (x, y)
```

בפועל, אין הבדל בין מתודות לפונקציות.

כאשר קוראים לפונקציה, למעשה קוראים למתודה של אובייקט כללי. בתוך פונקציה כזו, מילת המפתח this מתייחסת לאובייקט כללי, לכן, אין הבדל טכני בין מתודות לפונקציות.

ההבדל האמיתי הוא בעיצוב ובתכלית, מתודות נכתבות על מנת לפעול בצורה כלשהי על האובייקט, בעוד שפונקציות בד"כ עומדות בעד עצמן ואינן משתמשות באובייקט.

יצירת פונקציה:

```
function compute_area()
{
  return this.width * this.height;
}
```

יצירת אובייקט חדש מסוג -Rectangle שימוש ב ctor שהוגדר קודם:

```
var page = new Rectangle(8.5,11);
```

הגדרת מתודה ע"י הצבת הפונקציה לתוך תכונה של אובייקט:

```
page.area = compute_area();
```

קריאה למתודה החדשה:

```
var a = page.area();
```

לפני הקריאה למתודה area( ) עבור האובייקט rect יש להציב את המתודה כתכונה של האובייקט. לאחר הצבה זו, לא ניתן לקרוא למתודה עבור אובייקטים אחרים, לפני שהתבצעה ההצבה הזו עבורם.

שימוש ב Get ו Set

Get

שימוש ב get מאפשר לנהל את הערך המוחזר של תכונה, ע"י קישור פונקציה לתכונה של האובייקט. בעת הגישה לתכונה זו, תקרא למעשה הפונקציה המקושרת.

פונקצית get לא מקבלת פרמטרים, ושמה חייב להיות יחודי.

דוגמא לשימוש ב get באובייקט חדש:

```
var str = ['test'];
var obj = {
  get latest () {
    if (str.length == 0) return undefined;
    return str[str.length - 1]
  }
}
console.log (obj.latest); // יחזיר "test".
```

דוגמא לשימוש ב get באובייקט קיים:

```
var o = { a:0 }

Object.defineProperty(o, "b", { get: function () { return this.a + 1; } });

console.log(o.b)
```

הסרת פונקצית get:

```
delete obj.latest;
```

set

השימוש ב set נועד לנהל את הכנסת הערכים לתכונות של האובייקט ע"י ביצוע בדיקות תקינות ועוד.

דוגמא לשימוש ב set באובייקט חדש:

```
var o = {
  set SetBornDate (val) {
        if (val > 0 && val <=120)
                this.bornDate = val;
  },
bornDate: 0;
}
```

דוגמא לשימוש ב set באובייקט קיים:

```
var o = { a:0 };

Object.defineProperty(o, "b", { set: function (x) { this.a = x / 2; } });

o.b = 10; //
console.log(o.a) // 5
```

אב-טיפוס וירושה

לכל אובייקט יש אב-טיפוס, כאשר האובייקט יורש את כל תכונותיו ממנו. כלומר, כל התכונות של אובייקט האב-טיפוס יהוו תכונות של האובייקטים שיורשים אותו.

ניתן להגדיר prototype כאשר יוצרים constractor.

דוגמא:

הגדרת מתודת בנאי למחלקה:

```
function Circle(x,y,r)
{
   this.x = x;
   this.y = y;
   this.r = r;
}
```

יצירת אובייקט:

```
var c = new Circle(0,0,0);
```

הגדרת קבוע : תכונה שתהיה משותפת לכל האובייקטים

```
Circle.prototype.pi = 3
```

הגדרת מתודה לחישוב היקף המעגל:

```
function Circle_circumference()
{
  return 2* this.pi* this.r;
}
Circle.prototype.circumference = Circle_circumference;
```

הגדרת מתודה ע"י שימוש ב prototype:

```
Circle.prototype.area =
 new function("return this.pi * this.r * this.r;");
```

יצירת מופעים וקריאה למתודות:

```
var c = new Circle(0.0,0.0,1.0);
var a = c.area();
var p = c.circumference();
```

תכונות ומתודות של prototype שייכות לכל מופעי המחלקה והן מתאימות לשימוש בנתונים משותפים לכל האובייקטים במחלקה (בדומה למשתנים ופונקציות סטטיות).

לדוגמא, מתאים להגדיר קבוע לכל האובייקטים מסוג מחלקה זו:

```
Circle.prototype.pi = 3.14;
```

יתרון השימוש ב prototype:

1. השימוש באב-טיפוס יכול להפחית באופן משמעותי את כמות הזיכרון הדרושה עבור כל אובייקט, מכיוון שהוא יכול לרשת הרבה מתכונותיו.
2. אובייקט יורש תכונות אפילו אם התווספו לאב-טיפוס אחרי יצירת האובייקט.

שפה מונחית עצמים

בגרסאות הקודמות של JS, עד גרסת ECMA6, JS תומכת בסוג נתונים של אובייקטים, אין בה מושג פורמלי של מחלקה (class).

התפיסה המקובלת בשפות מונחות עצמים היא שהן תומכות בירושה של מחלקות, וכך גם JS. מצד שני JS ,עושה שימוש "כבד" באובייקטים, ויש לה סוג אב-טיפוס משלה מבוסס על ירושה.

JS היא בעצם כמו שפה מונחת עצמים, היא שואבת השראה משפות מונחות עצמים אחרות שמשתמשות באב טיפוס המבוסס על ירושה במקום מחלקות המבוססות על ירושה. למרות זאת, היא מצליחה לחקות את המאפיינים של שפות מבוססות מחלקות יורשות כמו java.

אובייקט הוא מבנה נתונים המכיל מספר של שמות נתונים ומתודות שפועלות עליהם. קבוצות של אובייקטים מיחסות ערכי נתונים ומתודות לתוך חבילה אחת, אשר בד"כ עושה את התכנות לקל יותר ע"י הגדלת המודולריות של הקוד.

בשפות מונחות עצמים כדוגמת java, מחלקה מגדירה את המבנה של האובייקט, התכונות והמתודות. ב js לא קיים המושג הפורמלי של מחלקה, אך ע"י פונקציות ה ctor ואובייקטי ה prototype התוצר הסופי דומה למחלקות.

אובייקטים ב JS יכולים להכיל מספר בלתי מוגבל של תכונות, ותכונות יכולות להתווסף לאובייקט באופן דינמי. בשפות מונחות עצמים מקובלות כל התכונות מוגדרות מראש, ולא ניתן להוסיף תכונות באופן דינמי. לכן, כאשר משתמשים באובייקטים ב JS ע"מ לחקות את התנהגות השפות מונחות העצמים נגדיר מראש את קבוצת התכונות עבור כל אובייקט וסוג הנתון של התכונה.

מרכיבי תכנות מונחה עצמים

תכונות

בדוגמא זו השדה r הינו תכונה.

```
function Circle(x,y,r)
{
   this.x = x;
   this.y = y;
   this.r = r;
}
```

מתודות

בדוגמא זו הפונקציה circumference היא מתודה.

```
function Circle_circumference()
{
  return 2* this.pi* this.r;
}
Circle.circumference = Circle_circumference;
```

משתנים סטטיים

משתנה הנוצר פעם אחת ויחידה בעת יצירת המופע הראשון של המחלקה, והוא משותף לכל המופעים מטיפוס המחלקה.

הגישה למשתנה סטטי דרך שם המחלקה ולא שם המופע.

דוגמא:

```
Circle.PI = 3.14;
```

מתודות סטטיות

מתודה ברמת המחלקה ולא ברמת המופע, מהווה פונקציית שירות של המחלקה.

הגישה למתודה סטטית ע"י שם המחלקה ולא שם המופע. השימוש ב this אינו רלוונטי.

דוגמא- המחלקה Circle:

יצירת ctor ותכונה:

```
function Circle(radius)
{
   this.r = radius;
}
```

משתנה סטטי:

```
Circle.PI = 3.14159;
```

פונקציה המחשבת שטח מעגל:

```
function Circle_area()
{ return Circle.PI * this.r; }
```

פונקציה המחזירה את העיגול הגדול מבין 2 עיגולים:

```
function Circle_max(a,b)
{
  if(a.r>b.r)
    return a;
  else
    return b;
}
```

המתודה Circle_max מבצעת פעולה על 2 אובייקטים מטיפוס Circle. לפיכך היא מתאימה להיות מתודה סטטית ולא מתודה הפועלת על מופע.

```
Circle.max = Circle_max;
var c = new Circle(1.0);

c.r = 2.2;
var a = c.area();
var x = Math.exp(Circle.PI);
var d = new Circle(1.2);

var bigger = Circle.max(c,d);
```

תכנות מונחה עצמים בגרסת ECMA6

הגדרת מחלקה:

```
class Animal{}
```

Constractor:

```
class Animal {
constructor(){
console.log("Animal created");
}
}
```

מתודות:

```
class Animal {

makeNoise(){
alert("animal says);
}
```

מתודה סטטית:

```
static addAnimal(){
```

```
if(!Animal.count)
Animal.count=0;
Animal.count++;
}
}
```

יצירת אובייקט מטיפוס המחלקה:

```
var anim = new Animal();
anim. makeNoise();
```

השמת אובייקט במערך:

```
var arr = [];
arr[0] = anim;
arr[1] = new Animal();
```

הגדרת ירושה:

```
class Cat extends Animal{}
```

קריאה ל ctor של מחלקת הבסיס:

```
class Cat extends Animal{
constructor(){
super();
this.name="Mitzi";
}
```

דריסת פונקציה של מחלקת הבסיס:

```
makeNoise(){
super.makeNoise();
console.log("Meawoo");
}
```

דוגמא מלאה למחלקות Animal ו Cat:

```

class Animal{

constructor(name) {
this._name = name;
Animal.addOne();
}

get name() {
return this._name.toUpperCase();
}

set name(value) {
this._name = value;
}

//הגדרת מתודה
```

```
makeNoise(){
console.log("animal says …");
}
static addOne(){
if(!Animal.count)Animal.count=0;
Animal.count++;
}

}


class Cat extends Animal{

constructor(name){
super(name);
this.houseTrained=false;
}
makeNoise(){
super.makeNoise();
alert("Meyawoooo");
}
}

//testing:
var c=new Cat();
c.makeNoise();
alert(“Total animals counter: ”+Animal.count);
```