

JavaScript Objects

אובייקטים - objects

לאובייקט יש פונקציות ומאפיינים לדוגמא

```
let person = {  
  firstName: 'John', lastName: 'Deer', getFullName: function () {  
    return this.firstName + this.lastName;  
  }  
};
```

ל `this` יש משמעויות רבות ב JS אבל בתוך פונקציה של אובייקט הוא מצביע על האובייקט שהפעיל את הפונקציה

```
alert(person.firstName);  
alert(person.getFullName());
```

גישה למאפיין או פונקציה של אובייקט נעשית בעזרת שם האובייקט ונקודה

בהגדרת האובייקט אפשר לרשום את `firstName`, `lastName` עם מרכאות

אובייקט וזיכרון

חשוב להבין שיש כאן שני חלקים :

- הקצאת מקום לאובייקט קרי כתובת - על ה stack
- הקצאת מקום לאיברים של האובייקט - על ה heap

JS מבצע את הקצאת זיכרון בצורה אוטומטית

JS מבצע את שחרור הזכרון בצורה אוטומטית בעזרת garbage collector

מערך של אובייקטים

ראה דוגמא [כאן](#)

reference type מול value type

Value type

- Number
- Boolean
- String
- Symbol(es6)

Reference type

- Object
- Array
- Function
- Date

הערך בתא הוא **הערך** של המשתנה
המשתנה מוקצה באזור זכרון שנקרא **stack**
זה זיכרון מהיר אך מאוד מוגבל במקום

הערך בתא הוא **הכתובת** אליה המשתנה מצביע.
המשתנה מוקצה ב **stack** והמידע עצמו באזור
זכרון שנקרא **heap** זה זיכרון פחות מהיר אבל בעל
הרבה יותר מקום ביחס ל **stack**

השוואה בין אובייקטים

להבין את מה משווים :

את הערך של הכתובת

או את מה שהאובייקט מצביע עליו

דוגמא 1 - Reference type vs value type

```
let a=1,b,obj1 = {name : 'Avi'},obj2;
```

```
b=a;
```

← השמת ערך !!!

```
a=2;
```

```
console.log(b,b==a);
```

← מה אתם מצפים לקבל ?

```
obj2=obj1;
```

← השמת כתובת !!!

```
obj1.name='Yossi';
```

```
console.log(obj2,obj1 == obj2);
```

← מה אתם מצפים לקבל ?

איך תראה תמונת הזיכרון הכוללת את
התאים a,b,obj1,obj2, **בלי ההבנה של**
זה לא תבינו מה קורה

דוגמא 2 - Reference type vs value type

```
let array1,array2 = [1,2,3,6];
```

```
array1 = array2;
```

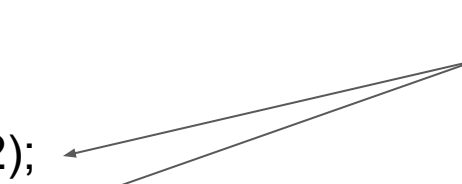
```
array1[0]=8;
```

```
array2[1]=18;
```

```
console.log(array1 == array2);
```

```
console.log(array1,array2)
```

מה אתם מצפים לקבל



איך תראה תמונת הזיכרון הכוללת את
התאים array1 , array2 , **בלי ההבנה**
של זה לא תבינו מה קורה

דוגמא 3 - Reference type vs value type

```
let array1 = [{ name: "Jim", age: 33 }, { name: "John", age: 22 }],
```

```
array2 = [1, 2, 3, 6],
```

```
a,
```

```
obj1;
```

```
a = array2[2];
```

```
a = 44;
```

```
console.log(array2);
```

```
obj1 = array1[1];
```

```
obj1.age = 55;
```

```
console.log(array1);
```

מה אתם מצפים לקבל

איך תראה תמונת הזיכרון הכוללת את
התאים array1 , array2 , a , obj1
בלי ההבנה של זה לא תבינו מה קורה

שימוש בפונקציה כמשתנה

```
const val = function (num1 , num2){  
  return num1+num2;  
}
```

```
console.log(val(1,2));
```

יצירת אובייקט - constructor function \ factory function

```
function Person (name ,age){  
  this.name = name;  
  this.age = age;  
  this.write = function(){  
    console.log(`name : ${this.name} , age : ${this.age}`)  
  }  
}
```

פונקצית הבנאי נכתבת בדרך כלל באות גדולה לדוגמא Person

לכל פונקציה יש אובייקט this אשר מצביע לאובייקט הגלובלי - window עבור הדפדפן , אבל new גורם לו להצביע אל האובייקט שנוצר

```
let jim = new Person('Jim' , 33);  
let jane = new Person('Jane' , 21);  
console.log(jim.name);  
jim.write();
```

האופרטור new יוצר אובייקט ריק ואז פונקצית הבנאי מופעלת. האובייקט שנוצר והאובייקט jim מצביעים לאותו מקום בזיכרון heap

ירושת אובייקט - prototype property

- לכל אובייקט JS יש מאפיין פרטי בשם **prototype** אשר מאפשר ירושה. המאפיין הזה מצביע אל אובייקט בשם prototype
- אובייקט יכול לשים תחת מאפיין prototype את כל המאפיינים והפונקציות שהוא מאפשר לרשת
- כל אובייקט יורש by default אובייקט בשם Object ויכול להשתמש בפונקציות שלו לדוגמא `hasOwnProperty`
- ירושה בין אובייקטים מתאפשרת באמצעות מושג שנקרא **prototype chain**
- שפות מונחות עצמים כמו C++ וכיוצא באלה תומכות בירושה של מחלקות ואילו JS תומך בירושת אובייקטים. בשני המקרים המטרה לעשות code reuse ז"א להשתמש בקוד קיים ולא לכתוב אותו מחדש

דוגמא לשימוש ב prototype לשיתוף מאפיין

```
var Person = function (name ,age){  
  this.name = name;  
  this.age = age;  
  this.write = function(){  
    console.log(`name : ${this.name} , age : ${this.age}`)  
  }  
}  
Person.prototype.nationality = "Israel";
```

Every object instantiated using new Person will have nationality as property. **Benefit : less memory consumption , less code to write**

```
let jim = new Person('Jim' , 33);  
let jane = new Person('Jane' , 21);
```

```
console.log(jim.nationality,jane.nationality);
```

Will console Israel Israel

Prototype in the console and prototype chain

Add `console.log(jim)` to previous sample and you can see the nationality under **__proto__** which stand for the prototype

```
▼ Person {name: "Jim", age: 33, write: f} ⓘ
  age: 33
  name: "Jim"
  write: f ()
  ▼ __proto__:
    nationality: "Israel"
    constructor: f (name ,age)
    ▼ __proto__:
      constructor: f Object()
      hasOwnProperty: f hasOwnProperty()
      isPrototypeOf: f isPrototypeOf()
      propertyIsEnumerable: f propertyIsEnumerable()
      toLocaleString: f toLocaleString()
      toString: f toString()
      valueOf: f valueOf()
      __defineGetter__: f __defineGetter__()
      __defineSetter__: f __defineSetter__()
      __lookupGetter__: f __lookupGetter__()
      __lookupSetter__: f __lookupSetter__()
      get __proto__: f __proto__()
      set __proto__: f __proto__()
```

Notice that there is another `__proto__` and this belong to `Object` which is inherited by ANY object default. This is called [prototype chain](#)

שימוש ב `__proto__` להצגת מאפיינים ופונקציות של מערך

```
let array = [1, 5, 8];  
console.log(array);
```

```
▼ Array(3)   
  0: 1  
  1: 5  
  2: 8  
  length: 3  
  __proto__: Array(0)  
    ▶ concat: f concat()  
    ▶ constructor: f Array()  
    ▶ copyWithin: f copyWithin()  
    ▶ entries: f entries()  
    ▶ every: f every()  
    ▶ fill: f fill()  
    ▶ filter: f filter()  
    ▶ find: f find()  
    ▶ findIndex: f findIndex()  
    ▶ flat: f flat()  
    ▶ flatMap: f flatMap()  
    ▶ forEach: f forEach()  
    ▶ includes: f includes()  
    ▶ indexOf: f indexOf()  
    ▶ join: f join()  
    ▶ keys: f keys()  
    ▶ lastIndexOf: f lastIndexOf()  
    length: 0  
    ▶ map: f map()  
    ▶ pop: f pop()
```

דוגמא לשימוש ב prototype לשיתוף פונקציה

```
function Person (name,age) {  
  this.name = name;  
  this.age = age;  
};
```

**Benefit : less memory
consumption , less
code to write**

```
Person.prototype.write = function() {  
  console.log(`name : ${this.name} , age : ${this.age}`);  
}
```

```
let person1 = new Person("Nathan" , 33);  
let person2 = new Person("Nitzan" , 29);  
person1.write();  
person2.write();
```

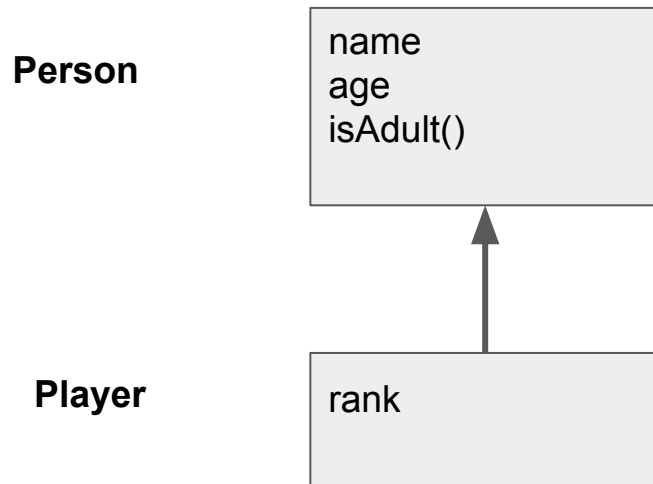

Create object using Object.create

```
var personProto = {  
  name : 'some person' , age : 22 , write : function(){  
    console.log(`name : ${this.name} , age : ${this.age}`)  
  }  
}
```

```
var jim = Object.create(personProto);  
jim.name = 'jim';  
jim.age = 33;  
jim.write();
```

פחות בשימוש מ
function constructor

דוגמא קלאסית לירושה - אבל מימוש מלא חסר



מימוש הדוגמא הקודמת

```
function Person(name ,age){  
  this.name = name;  
  this.age = age;  
}
```

```
Person.prototype.isAdult = function(){  
  return age >= 18;  
}
```

```
function Player(name ,age,rank){  
  this.rank = rank;  
  Person.call(this,name,age);  
  // TBD , how to inherit isAdult ??????????  
}
```

קל לממש את הירושה זאת
באמצעות class ונראה זאת בהמשך
ב react

isPrototypeOf

Checks if an object exist in another object :

```
const Animal = {  
  isAnimal: true  
};  
const Lion = Object.create(Animal);  
Lion.isLion = true;
```

```
console.log(Animal.isPrototypeOf(Lion)); //true  
console.log(Lion.isPrototypeOf(Animal)); //false  
console.log(Lion);  
console.log(Animal);
```

19:18:11.158 true

19:18:11.158 false

19:18:11.158

```
{isLion: true} ⓘ  
  isLion: true  
  __proto__: Object  
    isAnimal: true  
    proto : Object
```

19:18:11.159

```
{isAnimal: true} ⓘ  
  isAnimal: true  
  __proto__: Object
```

instanceof

The **instanceof operator** tests whether the prototype property of a constructor appears anywhere in the prototype chain of an object.

```
function Person(name,age){  
    this.name=name;  
    this.age = age;  
}  
  
const p = new Person('Nathan',44);  
let n=1;  
  
console.log(p instanceof Person); // true  
console.log(n instanceof Person); // false
```