ECE_ΓΚ802 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΔΙΑΔΙΚΤΥΟΥ
8ο εξάμηνο

JavaScript - μέρος 2
(κεφάλαιο 9)

Τύποι δεδομένων αναφοράς: Πίνακες, Συναρτήσεις και Αντικείμενα στη JavaScript

Διαφορές μεταξύ πρωτογενών δεδομένων και δεδομένων αναφοράς

1. Τα πρωτογενή δεδομένα είναι μη τροποποιήσημα (immutable), ενώ τα δεδομένα αναφοράς μπορούν να τροποποιηθούν.
2. Τα δεδομένα αναφοράς έχουν ιδιότητες κάτι που δεν έχουν τα πρωτογενή δεδομένα,
3. Η αντιγραφή ενός πρωτογενούς δεδομένου a σε μια νέα μεταβλητή (πχ. let b = a;) δημιουργεί νέο αντίγραφο της τιμής του πρωτοτεγενούς τύπου, ενώ στα δεδομένα αναφοράς δημιουργεί μια νέα αναφορά στο ήδη υπάρχον αντικείμενο.
4. Αυτό έχει ιδιαίτερη σημασία στις μεταβλητές που περνάμε στα ορίσματα συναρτήσεων.

1. πίνακες

Πίνακες: αντικείμενα τύπου Array

Οι πίνακες περιέχουν στοιχεία ως ακολουθία,
αναφερόμαστε στα στοιχεία με δείκτες 0 … array.length-1

```
const shopping = ['ψωμί', 'γάλα', 'τυρί', 'μήλα'];
const sequence = [1, 1, 2, 3, 5, 8, 13];
const random = ['tree', 795, [0, 1, 2]];
```

typeof shopping // 'object'

αναφορά σε στοιχεία πίνακα, μήκος πίνακα

```
const seq = [1, 1, 2, 3, 5, 8, 13];

seq[2] // δείκτης σε στοιχείο πίνακα

// επιστρέφει 2

seq[1] = 8; // εκχώρηση τιμής σε στοιχείο πίνακα

seq

// επιστρέφει [1, 8, 2, 3, 5, 8, 13]

seq.length;  // το πλήθος στοιχείων του πίνακα

// επιστρέφει 7
```

Δημιουργία πίνακα μέσω **literal** ή **new Array()**

δύο ισοδύναμοι τρόποι: μέσω literal ή με κλήση της συνάρτησης δημιουργού αντικειμένου Array()

```
let c = [10,20];
let c = new Array(10,20);  //array με δύο στοιχεία
```

new Constructor-function : τελεστής δημιουργίας αντικειμένου μέσω συνάρτησης-δημιουργού (Array() είναι η συνάρτηση-δημιουργός πινάκων)

new Array()

Όταν η συνάρτηση-δημιουργός Array() έχει όρισμα έναν ακέραιο, αυτός ορίζει το μέγεθος του πίνακα

```
let c = new Array(); //μη ορισμένου μεγέθους
let c = new Array(2); // δημιουργεί άδειο πίνακα με length = 2
```

πώς γεμίζουμε ένα πίνακα;

```
const arr = new Array(5).fill(5);
arr
```

[5, 5, 5, 5, 5]

```
const arr2 = [...Array(5)]
arr2
```

[undefined, undefined, undefined, undefined, undefined]

παράδειγμα

```
f = new Array("5");
g = new Array(5);
```

ποιο το αποτέλεσμα των παρακάτω;

```
f.length
> 1

g.length
> 5
```

Αρχικοποίηση με βρόχο επανάληψης

```
// δημιουργία δύο νέων πινάκων
let n1 = new Array( 5 ); // πίνακας 5 στοιχείων
let n2 = new Array(); // άδειος πίνακας

// αρχικοποίηση πίνακα με γνωστό μέγεθος
for ( let i = 0; i < n1.length; ++i )
   n1[ i ] = i;

// αρχικοποίηση πίνακα χωρίς καθορισμένο μέγεθος
for ( i = 0; i < 5; ++i )
   n2[ i ] = i;
```

n.length το πλήθος των στοιχείων του πίνακα

Αραιοί πίνακες

Μπορούμε να εισάγουμε στοιχεία πέραν του μήκους τους

```
const ar = [1,2,3];
undefined
ar[10] = 20;
20
console.log(ar)
(11) [1, 2, 3, empty × 7, 20]
ar.length
11
```

τελεστής const
σε ορισμό πίνακα

```
> const a = [];
> a.push(1);
> a.push(5,6);
> a
 [1, 5, 6]
```

σημείωση:
ένα αντικείμενο μπορεί να οριστεί με τον τελεστή const, όμως το περιεχόμενο του μπορεί να τροποποιηθεί

2. μέθοδοι πινάκων

προσθήκη/αφαίρεση στοιχείων από το τέλος του πίνακα

Πίνακας = σωρός (stack)

Η μέθοδος **pop** καταργεί το τελευταίο στοιχείο του πίνακα. Το στοιχείο που καταργήθηκε επιστρέφεται.

Η μέθοδος **push** προσθέτει ένα νέο στοιχείο στο τέλος του πίνακα. Το νέο μήκος του πίνακα επιστρέφεται.

pop, push(element)

```
let myArray = ["Athens", "Patras", "Thessaloniki", "Volos"]
myArray.push('Larissa'); // επιστρέφει 5
myArray.push('Katerini', 'Kavala'); // επιστρέφει 7
myArray.pop(); // επιστρέφει "Kavala"
```

προσθήκη/αφαίρεση στοιχείων από την αρχή του πίνακα

Η μέθοδος **shift()** αφαιρεί ένα στοιχείο από την αρχή του πίνακα. Το στοιχείο του πίνακα επιστρέφεται.

Η μέθοδος **unshift(στοιχείο)** προσθέτει ένα νέο στοιχείο στην αρχή του πίνακα. Το νέο μήκος του πίνακα επιστρέφεται.

shift() unshift(element)

```
let myArray = ["Athens", "Patras", "Thessaloniki", "Volos"]
myArray.unshift('Larissa'); // επιστρέφει 5
myArray.unshift('Katerini', 'Kavala'); // επιστρέφει 7
myArray.shift(); // επιστρέφει "Katerini"
```

παράδειγμα

```
myArray
(4) ["Larissa", "Patras", "Thessaloniki", "Volos"]

myArray.pop()
"Volos"

myArray.shift()
"Larissa"
```

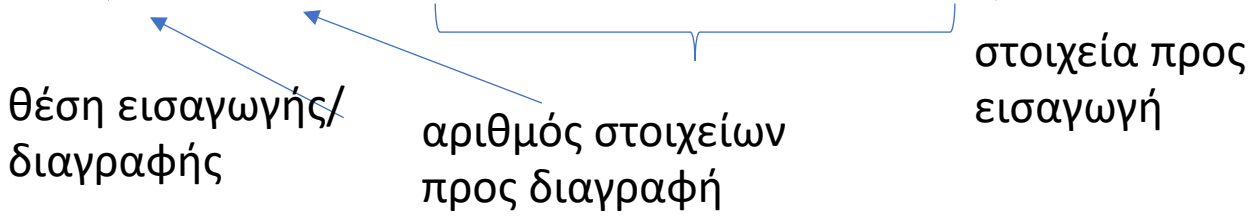
από string σε array **split()**
από array σε string **join()**

```
let myData = 'Athens, Patras, Thessaloniki, Volos';
let myArray = myData.split(','); //
(4) ["Athens", "Patras", "Thessaloniki", "Volos"]
        0: "Athens"
        1: "Patras"
        2: "Thessaloniki"
        3: "Volos"
let myNewString = myArray.join(' ');
myNewString
"Athens Patras Thessaloniki Volos"
```

splice: εισαγωγή και διαγραφή στοιχείων σε πίνακα

splice(θέση, πλήθος-διαγραφή, στοιχεία-για-εισαγωγή)

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```



αποτέλεσμα:

```
Banana,Orange,Lemon,Kiwi,Apple,Mango
```

splice(n)

Αν η splice πάρει μόνο ένα όρισμα, τότε θεωρούμε ότι τα στοιχεία για διαγραφή εκτείνονται από τη θέση n μέχρι το τέλος του πίνακα, χωρίς στοιχεία για εισαγωγή

```
const ar = [1,2,3,4,5,6]
let x = ar.splice(3);
console.log(ar);
[1,2,3]
```

άσκηση

Ποιο το αποτέλεσμα;

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 1);
console.log(fruits)
```

Banana, Orange, Mango

άσκηση

Ποιο το αποτέλεσμα του:

```
let fruits =
["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2); // διαγράφει από index..
console.log(fruits)
```

Banana, Orange

άσκηση

Ποιο το αποτέλεσμα του:

```
const ar = [1,2,3,4,5,6]
let x = ar.splice(3,2,30,40);
console.log(ar);
```

[1,2,3,30,40,6]

Προσθήκη στοιχείου σε array

```
const a=[1,"10", 3.14];
a[10]='kostas’;
console.log(a)
```

ποιο το αποτέλεσμα;

```
[1, "10", 3.14, empty × 7, "kostas"]
```

προσοχή: στην Python θα παίρναμε indexError

Μέθοδοι αναζήτησης

`indexOf()` και `lastIndexOf()`

- Επιστρέφουν τη θέση του στοιχείου που έχει τιμή === το όρισμα, η indexOf() αρχίζοντας από την αρχή, και η lastIndexOf() από το τέλος του πίνακα.
- Αν το στοιχείο δεν βρεθεί, επιστρέφουν την τιμή -1.

```
const ar = [5,10,15,20,10,5];
ar.indexOf(15);
> 2
ar.lastIndexOf(10);
> 4
```

ταξινόμηση πίνακα

```
let fruits =
["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
```

ταξινομεί τα στοιχεία αλφαβητικά, μπορούμε όμως να περάσουμε ως όρισμα της sort() μια συνάρτηση ταξινόμησης που ορίζει τη σειρά ανάλογα με την τιμή που επιστρέφει

```
let points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
```

παράδειγμα ταξινόμησης

```
<!DOCTYPE html>
<html>
   <head>
      <title>Ταξινόμηση πίνακα</title>
      <meta charset="utf-8">
      <script>
         var a = [ 10, 1, 9, 2, 8, 3, 7, 4, 6, 5 ];
         document.writeln( "<h1>Ταξινόμηση πίνακα</h1>" );
         outputArray( "Αρχική ακολουθία: ", a );
         outputArray( "Ταξινόμηση string: ", a.sort() );
         outputArray( "Σε αύξουσα σειρά: ", a.sort( comparePositive ) );
         outputArray( "Σε φθίνουσα σειρά: ", a.sort( compareNegative ) );
         function outputArray( heading, theArray )
           {document.writeln( "<p>" + heading +
                 theArray.join( " " ) + "</p>" );
           }
         function comparePositive( value1, value2 )
           {return parseInt( value1 ) - parseInt( value2 );
           }
         function compareNegative( value1, value2 )
           {return parseInt( value2 ) - parseInt( value1 );
           }
      </script>
   </head><body></body>
</html>
```

js arrays : σύγκριση με λίστες της python

JAVASCRIPT	PYTHON
a.length	len(li)
a.push(item)	li.append(item)
a.pop()	li.pop()
a.shift()	li.pop(0)
a.unshift(item)	li.insert(0, item)
a.slice(start, end)	li[start:end]
a.splice(start, howMany, […])	

άσκηση

```
const a = [1,2,3,4,5,6,7]
let b  = a.slice(3)
ποια η τιμή της a και b?
```

```
const a = [1,2,3,4,5,6,7]
let b  = a.splice(3)
ποια η τιμή της a και b?
```

2. Συναρτήσεις JS

Συναρτήσεις

- Η JavaScript είναι ισχυρά συναρτησιακή γλώσσα, χειρίζεται τις συναρτήσεις ως αντικείμενα (callable objects)
- Το **όνομα μιας συνάρτησης** θα πρέπει απαραίτητα να ξεκινάει από ένα αλφαβητικό χαρακτήρα (κεφαλαία ή πεζά) ή το χαρακτήρα της κάτω παύλας (underscore) ή $ (dollarsign)
- Εξ ορισμού, οι τιμές των **ορισμάτων εισόδου** αποδίδονται στον κώδικα της συνάρτησης με αντιγραφή τους σε τοπικές μεταβλητές (κλήση μέσω τιμής – by value)
- Αν το όρισμα εισόδου είναι κάποιος **πίνακας** ή άλλο **αντικείμενο**, αυτό αποδίδεται στην εσωτερική τοπική μεταβλητή μέσω αναφοράς (by reference)

συναρτήσεις functions

- Οι συναρτήσεις που ορίζονται εντός αντικειμένων objects ονομάζονται μέθοδοι **methods**
- Υπάρχει η δυνατότητα ορισμού **ανώνυμων συναρτήσεων** (αντίστοιχες των συναρτήσεων lambda της Python), ως ορίσματα άλλων συναρτήσεων.

ορισμός συνάρτησης με τη λέξη-κλειδί function

```
function onomaSynarthshs (orismata) {
  εντολές
  return values;
}
```

```
function random(number) {
  // τυχαίος αριθμός μεταξύ 1 και number
  return Math.floor(Math.random()*number+1);
}
let x = random(5);  //κλήση συνάρτησης
```

Αν μια συνάρτηση δεν περιέχει εντολή `return` στο τέλος εκτέλεσής της, τότε επιστρέφει την τιμή `undefined`.

const f = function()

Function Literal Notation
Εναλλακτικός τρόπος σύνταξης συνάρτησης - οφείλει την ύπαρξη της στο γεγονός ότι η JavaScript αντιμετωπίζει και τις συναρτήσεις ως αντικείμενα.

Μετά τη δήλωση συνάρτησης, αυτή μπορεί να κληθεί βάσει του ονόματος της.

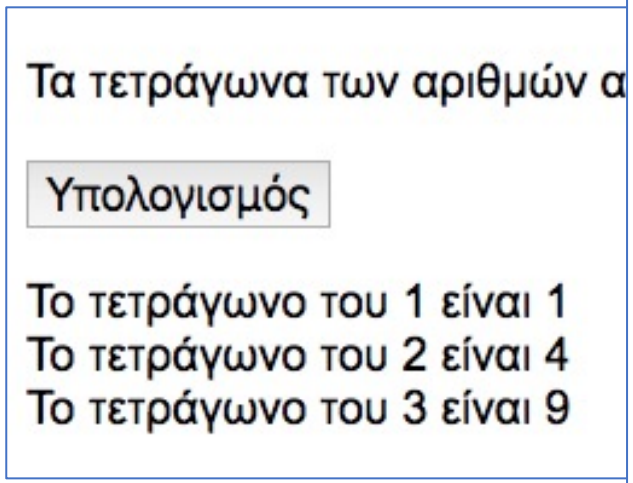
```
const όνομα_συνάρτησης =
 function( ορίσματα ) {
    //κώδικας
    return δεδομένα;
}
```

σε αυτή την περίπτωση δεν μπορούν να χρησιμοποιηθούν οι συναρτήσεις πριν τον ορισμό τους (όπως οι μεταβλητές).

Άσκηση

Να υπολογίσετε τα τετράγωνα των αριθμών από 1 έως x με κλήση συνάρτησης calculate() και square() που ορίζετε εσείς

```
<html>
<head>
<title>συνάρτηση square</title>
<style>
body {
font-family: Arial, Helvetica, sans-serif;
}
input, button, p {
font-size: 0.8em;
}
</style>
```




```
const theSquares = document.querySelector("squares");

function calculate(){
    theSquares.innerHTML = "";
    let range = document.querySelector("range").value;
    for ( let x = 1; x <= parseInt(range); x++ ) {
       theSquares.innerHTML += "Το τετράγωνο του " + x +
       " είναι " + square( x ) + "<br>";
    }
}

function square( y )
{
return y * y;
} // end function square
```

ex47-func-square.html

ES6 arrow functions

```
const x = (name) => {
    console.log('hi '+name);
}
//Παράδειγμα χρήσης
x('Nikos')
hi Nikos
```

Αν η συνάρτηση περιέχει μόνο μια εντολή return, απλοποιείται σε (val) => έκφραση

```
let x = (z) => z2;

x(5)
25
```

Ορισμός συνάρτησης χειριστή συμβάντων

```
const f = () => alert("clicked!");

document.querySelector("div").onclick = f;
```

προσοχή όχι f()

```
document.querySelector("div").onclick =
        () => alert("clicked!");
```

συνάρτηση => επιστρέφει αντικείμενο {... }

Έστω ότι συνάρτηση επιστρέφει αντικείμενο:
{name:"Κώστας", age:20}

```
const f = (n, a) => {name:n, age:a}
```

SyntaxError:

```
const f = (n, a) => ({name:n, age:a})
```

Η λέξη **this**

Η λέξη this παίρνει διαφορετικές τιμές ανάλογα με το πού χρησιμοποιείται:

- Σε μια μέθοδο, αναφέρεται στο αντικείμενο ιδιοκτήτη της μεθόδου.
- Σε μια συνάρτηση, αναφέρεται στο αντικείμενο global.
- Σε μια συνάρτηση, σε *strict mode*, είναι undefined.
- Σε event, αναφέρεται στο στοιχείο που σχετίζεται με το event.

Η λέξη **this** παράδειγμα

```
<html>
<body>
   <p>Παράδειγμα της λέξης this</p>
   <button onclick="myfunc(this)">αλλαγή χρώματος</button>
   <script>
      function myfunc(element){
            element.style.backgroundColor='yellow';
      }
   </script>
</body>
</html>
```

Εμφώλευση συναρτήσεων

Επιτρέπεται να οριστεί μια συνάρτηση μέσα σε μια άλλη συνάρτηση. Η εμφωλευμένη συνάρτηση μπορεί να κληθεί μόνο μέσα στην συνάρτηση που έχει οριστεί, και έχει πρόσβαση στις μεταβλητές της συνάρτησης αυτής.

```
function ypotinousa(a, b) {
   const sq = (x) => x*x;
   return Math.sqrt(sq(a) + sq(b));
}

ypotinousa(3,4)
> 5
```

Εμβέλεια μεταβλητών

- Έξω από όλες τις συναρτήσεις ορίζεται περιοχή καθολικής εμβέλειας **global scope**. Μεταβλητές που ορίζονται στο επίπεδο αυτό, είναι προσβάσιμες από όλον τον κώδικα.
- Προσοχή όλος ο κώδικας της σελίδας ανεξάρτητα από ποιο αρχείο ή <script> ανήκει, έχει πρόσβαση.
- Κάθε συνάρτηση ορίζει τοπική εμβέλεια **local scope** και οι μεταβλητές που ορίζονται στο επίπεδο αυτό είναι προσβάσιμες μόνο μέσα στη συνάρτηση

παράδειγμα

```
let x = 1;

function a() {
   let y = 2;
   b(x);
   b(y);
}

function b(value) {
     console.log( `Τιμή: ${value}`);
}
a();
```

Ποιο το αποτέλεσμα;
θα αλλάξει κάτι αν αντί για let χρησιμοποιήσουμε var;

```
> 'Τιμή: 1'
> 'Τιμή: 2'
```

τοπικές-καθολικές μεταβλητές

```
function myFunction() {
   let userName = 'Nikos';
   console.log(userName);
}
console.log(userName);
```

Ποιο το αποτέλεσμα;
θα αλλάξει κάτι αν αντί για let χρησιμοποιήσουμε var;

ReferenceError: userName is not defined, δεν θα αλλάξει ούτε με χρήση var αντί για let

τοπικές-καθολικές μεταβλητές

```
let color = 'μπλε';

function changeColor() {
    let anotherColor = 'κόκκινο';

    function swapColors() {
    let tempColor = anotherColor;
    anotherColor = color;
    color = tempColor;
     }
     swapColors();
}

changeColor();

console.log(color);
```

Ποιο το αποτέλεσμα;

> 'κόκκινο'

η ανταλλαγή χρωμάτων μεταξύ των μεταβλητών color και anotherColor θα γίνει κανονικά, η swapColor έχει πρόσβαση και στις 2 μεταβλητές

εκτέλεση JS σε strict mode

"use strict"; //ορίστηκε μετά την ES5

Πιο αυστηρή έκδοση της JS που δεν επιτρέπει κάποιες "χαλαρές" λειτουργίες, όπως η χρήση μεταβλητών που δεν έχουν δηλωθεί.

```
"use strict";
x = 3.14; // σφάλμα γιατί η x δεν έχει δηλωθεί
```

άσκηση

```
function myBigFunction() {
    let myValue = 1;
    subFunction1();
    subFunction2();
}
function subFunction1() {
    console.log(myValue);
}
function subFunction2() {
    console.log(myValue);
}
myBigFunction()
```

Ποιο το αποτέλεσμα;

ReferenceError:
myValue is not defined

Εμβέλεια συναρτήσεων. διαφορά let/var

```
function printing(){
      for(var i = 0; i<10; i++) {
            console.log(i)
      }
      console.log(i)
}
printing()
console.log(i)
```

```
1
2
3
4
5
6
7
8
9
10
Uncaught ReferenceError: i is not defined
```

Ποιο το αποτέλεσμα; ποια η διαφορά αν

```
for(let i = 0; i<10; i++)
```

Προαιρετικά ορίσματα συναρτήσεων

Προαιρετικά ορίσματα συναρτήσεων

Η JavaScript μέχρι την έκδοση ES6 δεν υποστήριζε άμεσα προαιρετικά ορίσματα με εξ ορισμού τιμές.

Λύση ήταν ο εσωτερικός έλεγχος με επιπλέον κώδικα που εντοπίζει τη μη χρήση του ορίσματος και του αναθέτει μια τιμή.

```
function myOwnFunction( optional ) {
    if ( typeof optional == 'undefined' )
    optional = 1;
    return optional + 1;
}

myOwnFunction(3); // Επιστρέφεται 4
myOwnFunction(); // Επιστρέφεται 2
```

```
optional = optional || 1;
```

Προαιρετικά ορίσματα συναρτήσεων στην ES6

Στην έκδοση ES6 επιτρέπονται πλέον προαιρετικά ορίσματα με προκαθορισμένες τιμές

```
function myOwnFunction( optional=1 ) {
    return optional + 1;
}

myOwnFunction(3); // Επιστρέφεται 4
myOwnFunction(); // Επιστρέφεται 2
```

Ο τελεστής ανάπτυξης `...` στα ορίσματα συνάρτησης

Ο τελεστής `... ` (spread) στα ορίσματα συναρτήσεων, μπορεί να χρησιμοποιηθεί για να κάνει τα στοιχεία ενός πίνακα ξεχωριστά γνωρίσματα, όταν κάτι τέτοιο απαιτεί ο ορισμός της συνάρτησης

```
function f(x,y,z){
return x+y+z;
}
const a = [5,6,7];
console.log(f(...a));
>18
```

Ο τελεστής ανάπτυξης `...` στα ορίσματα συνάρτησης

Ο τελεστής `...` (τελεστής ανάπτυξης, spread), είναι ένας τελεστής που χρησιμοποιείται για την ανάπτυξη ενός πίνακα η μιας ακολουθιακής δομής όπως μια συμβολοσειρά, στα επί μέρους στοιχεία από τα οποία απαρτίζεται.

παράδειγμα

```
const a = [1,2,3];
const b = [ ...a];
```

Ο τελεστής ανάπτυξης `...` στον ορισμό συνάρτησης

Μια άλλη περίπτωση χρήσης του τελεστή ανάπτυξης `...` είναι κατά τον ορισμό συνάρτησης με μεταβλητό πλήθος στοιχείων.

Στην περίπτωση αυτή ο τελεστής εφαρμόζεται σε μια μεταβλητή, την οποία μπορούμε στο σώμα της συνάρτησης να χειριστούμε ως πίνακα τιμών.

παράδειγμα

```
function max(...args) {
        let maxValue = args[0];
        for (let n of args) {
                if (n > maxValue) maxValue = n;
        }
        return maxValue;
}

max(10, 50, 60, 5, 8);
> 60
```

Στατικές μεταβλητές συνάρτησης

```
function counter() {
      return ++counter.count;
}
counter.count = 0;

counter(); // επιστρέφει 1
counter(); // επιστρέφει 2
```

Δεν θα πρέπει να ξεχνάμε ότι μια συνάρτηση είναι στην ουσία ένα object. Άρα μπορεί να έχει ιδιότητες, όπως όλα τα αντικείμενα.

Χρήση συνάρτησης για ορισμό μοναδικού χώρου ονομάτων

Για να πετύχουμε την απομόνωση του χώρου ονομάτων μεταβλητών (namespace) του κώδικά μας, μπορούμε να ενσωματώσουμε τον κώδικα σε μια συνάρτηση, την οποία να καλέσουμε μόλις το DOM φορτωθεί, και μέσα στη συνάρτηση αυτή να ορίσουμε τις μεταβλητές, συναρτήσεις, αντικείμενα κλπ.

```
function pageScript(){

    // εδώ ορίζουμε μεταβλητές, συναρτήσεις,
    // χειριστές γεγονότων, κλπ
}

document.addEventListener("DOMContentLoaded", pageScript);
```

Μέθοδοι επεξεργασίας πινάκων

Μέθοδοι επεξεργασίας πινάκων

* myArray.**forEach**(myFunction) εφαρμόζει τη συνάρτηση σε κάθε στοιχείο του πίνακα

* myArray.**map**(myFunction) δημιουργεί νέο πίνακα με εφαρμογή της συνάρτησης σε κάθε στοιχείο του

* myArray.**filter**(myFunction) δημιουργεί νέο πίνακα με τα στοιχεία που ικανοποιούν τη συνάρτηση φίλτρο

* myArray.**reduce**(acumFun, αρχικήΤιμή) παράγει μια τιμή μετά από διαδοχική εφαρμογή της συνάρτησης acumFun στα στοιχεία του πίνακα.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

myArray.**forEach**(f)

εφαρμόζει τη συνάρτηση f σε κάθε στοιχείο του πίνακα

Η συνάρτηση f παίρνει ως ορίσματα το εκάστοτε στοιχείο el, και προαιρετικά το indx του στοιχείου και το array

```
const myAr = ["Χίος", "Μυτιλήνη", "Σάμος"]
myAr.forEach((el, i) =>
     console.log(i, 'Η ' + el));
```

```
> 0 'Η Χίος'
> 1 'Η Μυτιλήνη'
> 2 'Η Σάμος'
```

myArray.**forEach**(myFunction)

Η forEach δεν τροποποιεί τον ίδιο τον πίνακα, όμως κάτι τέτοιο είναι δυνατό:

```
const myAr = ['Χίος', 'Μυτιλήνη', 'Σάμος'];
myAr.forEach((el, i, a) => {
     a[i] = a[i].toUpperCase();
});
console.log(myAr);

> [ 'ΧΊΟΣ', 'ΜΥΤΙΛΉΝΗ', 'ΣΆΜΟΣ' ]
```

ορίσματα (el, [indx, array])

myArray.**map**(f)

let ar2 = ar1.**map**((el, i, ar) => {
return … }

επιστρέφει ένα νέο πίνακα με εφαρμογή της συνάρτησης σε κάθε στοιχείο του αρχικού πίνακα

```
const myAr = ["Χίος", "Μυτιλήνη", "Σάμος"]

const newAr = myAr.map((el) => "Η νήσος "+el);
```

```
newAr
(3) ["Η νήσος Χίος", "Η νήσος Μυτιλήνη", "Η νήσος Σάμος"]
```

myArray.**filter**(f)

επιστρέφει νέο πίνακα με τα στοιχεία που ικανοποιούν τη συνάρτηση φίλτρο

```
let myAr = ["Χίος", "Μυτιλήνη", "Σάμος"]

let newAr = myAr.filter((el) => el.length > 5);


newAr
["Μυτιλήνη"]
```

άσκηση

Να διαγράψετε τα διπλά στοιχεία ενός πίνακα με χρήση της μεθόδου filter (διαγραφή των στοιχείων που έχουν index διάφορο της θέσης τους)

```
const myArray = [1, 1, 2, 2, 3, 4, 5, 7, 7];

let x = myArray.filter((el, ind, ar) => {
     return ar.indexOf(el) === ind;
});
```

myArray.**reduce**(accumFun, αρχικήΤιμή)

επιστρέφει μια τιμή μετά από διαδοχική εφαρμογή της συνάρτησης στα στοιχεία
Προσοχή: η συνάρτηση accumFun παίρνει ως ορίσματα(acum, el, ind, ar), το δεύτερο όρισμα της reduce αρχικήΤιμή, είναι η αρχική τιμή του acum

```
let myAr = ["Χίος", "Μυτιλήνη", "Σάμος"]

let result = myAr.reduce((islands, el) => {
islands += " "+ el}, "");
```

result *"Χίος Μυτιλήνη Σάμος"*

άσκηση

Να βρείτε το άθροισμα των στοιχείων ενός πίνακα

```
const myArray = [1, 1, 2, 2, 3, 4, 5, 7, 7];

let mySum = myArray.reduce((ac, el) => {
  return (ac += el);
});
console.log(mySum); // τυπώνει 32
```

άσκηση

Να τυπώσετε τα τετράγωνα των στοιχείων ενός πίνακα με 4 διαφορετικούς τρόπους

```
let ar = [1, 2, 3, 4, 5];

for (let _i = 0; _i < ar.length; _i++)
      console.log(ar[_i]2);

for (let i in ar) console.log(ar[i]2);

for (let x of ar) console.log(x2);

ar.forEach(x => console.log(x2));
```

άσκηση

Να βρείτε την ελάχιστη τιμή των στοιχείων του πίνακα:

```
const a = [5, 10, 18, 32, 20, 44];
```

```
const result = a.reduce((s, el) =>
    (el < s ? el : s), Infinity);
```

άσκηση

Να βρείτε την τυπική απόκλιση SD

```
const a = [5, 10, 18, 32, 20, 44];
```

$$\mathrm{SD} = \sqrt{\frac{\sum |x - \mu|^2}{N}}$$

3. αντικείμενα στη JS

αντικείμενα (objects) της javascript

Τα **objects** είναι δομές που περιέχουν ζεύγη **ιδιότητας/τιμής** (property/value).

Οι **ιδιότητες** μπορεί να πάρουν ως τιμή σταθερές, συναρτήσεις (που ονομάζονται μέθοδοι), ή άλλα αντικείμενα

object literal syntax

```
const car = {
    make: "volvo",
    speed: 140,
    engine: {
        size:1800,
        fuel: "diesel",
        pistons: ["piston1", "piston2"]},
    drive: function() { return "drive …"} }
```

δημιουργία αντικειμένου

```
const person1 = {
        firstName: "Γιάννης",
        lastName: "Ιωάννου",
        age: 50,
        eyeColor: "πράσινα",
        greeting: function() {
                alert('Γεια! με λένε ' + this.lastName + '.'); } }
```

Προσοχή: διαφορετική συμπεριφορά του **this** σε συναρτήσεις βέλη =>

```
greeting: () =>() {
  alert('Γεια! με λένε ' + this.lastName + '.');}
```

Αναφορά στις ιδιότητες ενός αντικειμένου

Έστω ότι θέλουμε την ιδιότητα firstName του αντικειμένου person1. Μπορούμε με δύο τρόπους:

```
person1.firstName
person1["firstName"]
```

Αντίστοιχα αναφορά στη μέθοδο greeting()

```
person1.greeting()
person1["greeting"]()
```

Άσκηση

```
const car = {
   make: "volvo",
   speed: 140,
   engine: {
      size:1800,
      fuel: "diesel",
      pistons: ["piston1", "piston2"]},
   drive: function() { return "drive …"} }
```

Πώς θα αναφερθούμε στο καύσιμο fuel;

car.engine.fuel ή car['engine']['fuel']

αντικείμενα στην JavaScript

Τα αντικείμενα της JavaScript δεν είναι απλά πίνακες αντιστοίχισης κλειδιών–τιμών.

Κάθε αντικείμενο συνοδεύεται από μια πρόσθετη ιδιότητα που έχει την τιμή `prototype` που είναι ένα αντικείμενο, του οποίου κληρονομεί τις ιδιότητες.

Αυτή είναι μια ιδιαιτερότητα της JavaScript που δεν συναντάται σε άλλες γλώσσες προγραμματισμού. Ο μηχανισμός αυτός λέγεται "*κληρονομικότητα μέσω πρωτοτύπου*".

Αντικείμενα

Στα αντικείμενα των οποίων οι ιδιότητες είναι απαριθμήσιμες (enumerable), μπορούμε να εφαρμόσουμε ένα βρόχο **for/in** ως εξής:

```
for (property in car) {
     console.log(property, car[property]);
}
```

```
> 'make' 'volvo'
> 'speed' 140
> 'engine' { size: 1800, fuel: 'diesel', pistons: [
'piston1', 'piston2' ] }
> 'drive' ƒ drive()'make'
```

ιδιότητες ενός αντικειμένου – πίνακας

Τις ιδιότητες ενός αντικειμένου (μόνο τις enumerable) μπορούμε να τις ανακτήσουμε μέσω της μεθόδου `Object.keys()`

```
Objec.keys(car);
> [ 'make', 'speed', 'engine', 'drive' ]
```

Επίσης θα πρέπει να αναφερθεί ότι και ένας πίνακας έχει ως ιδιότητες τους δείκτες 0,1,2 ..
άρα:

```
const ar = [10, 20, 30];
Object.keys(ar);
> [ '0', '1', '2' ]
```

Μετατροπή αντικειμένων σε JSON

Η μετατροπή ενός αντικειμένου JS σε μια συμβολοσειρά από την οποία εν συνεχεία μπορεί να ανακτηθεί λέγεται **σειριοποίηση** (serialization).

Η JSON (JavaScript Object Notation), στην οποία μπορούμε να μετατρέψουμε τα αντικείμενα της JavaScript κατά τη σειριοποίησή τους είναι ένα πρότυπο ανταλλαγής δεδομένων με ευρεία χρήση, πέραν της JavaScript.

Η μετατροπή ενός αντικειμένου σε μορφή JSON γίνεται με τη μέθοδο `JSON.stringify(obj)`, ενώ η αντίθετη μετατροπή γίνεται με τη μέθοδο `JSON.parse(st)`

Αντικείμενα Date, συναρτήσεις, κανονικές εκφράσεις, και αντικείμενα τύπου Error, καθώς και τιμές undefined δεν μπορούν να μετατραπούν σε JSON

Δημιουργία κλάσεων και αντικειμένων

object literals

```
const ob = {};
```

```
> ob
{}
__proto__: Object
```

δημιουργία αντικειμένου με τη λέξη **new**

constructor function syntax

```
function Person(name) {
  this.name = name;
  this.greeting = function() {
    alert('Γεια με λένε'+ this.name + '.');
  };
}
```

```
let p1 = new Person('Bob');
```

Η κλήση της **new Person()** δημιουργεί αντικείμενα

Δημιουργία κλάσεων με new δημιουργός()

```
function Car(make, speed){
    this.make = make;
    this.speed = speed;
}
```

```
const myCar = new Car("VW", 200);
> myCar
Car {make: "VW", speed: 200}
   make: "VW"
   speed: 200
   __proto__:
      constructor: ƒ Car(make, speed)
      __proto__: Object
```

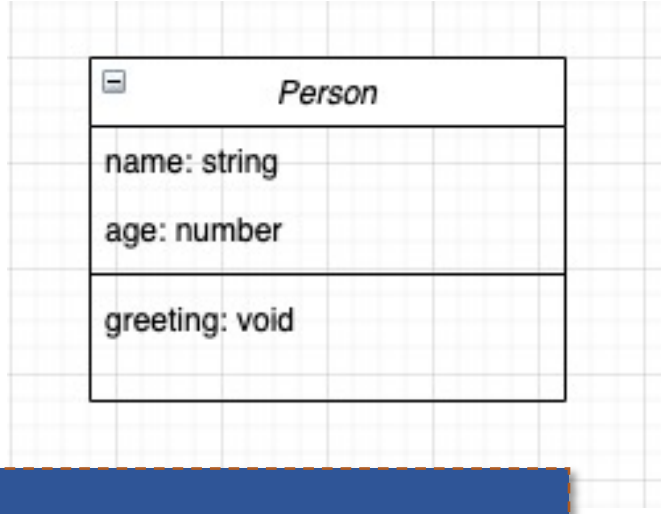
ορισμός μεθόδου στην κλάση, μέσω του πρωτοτύπου της κλάσης

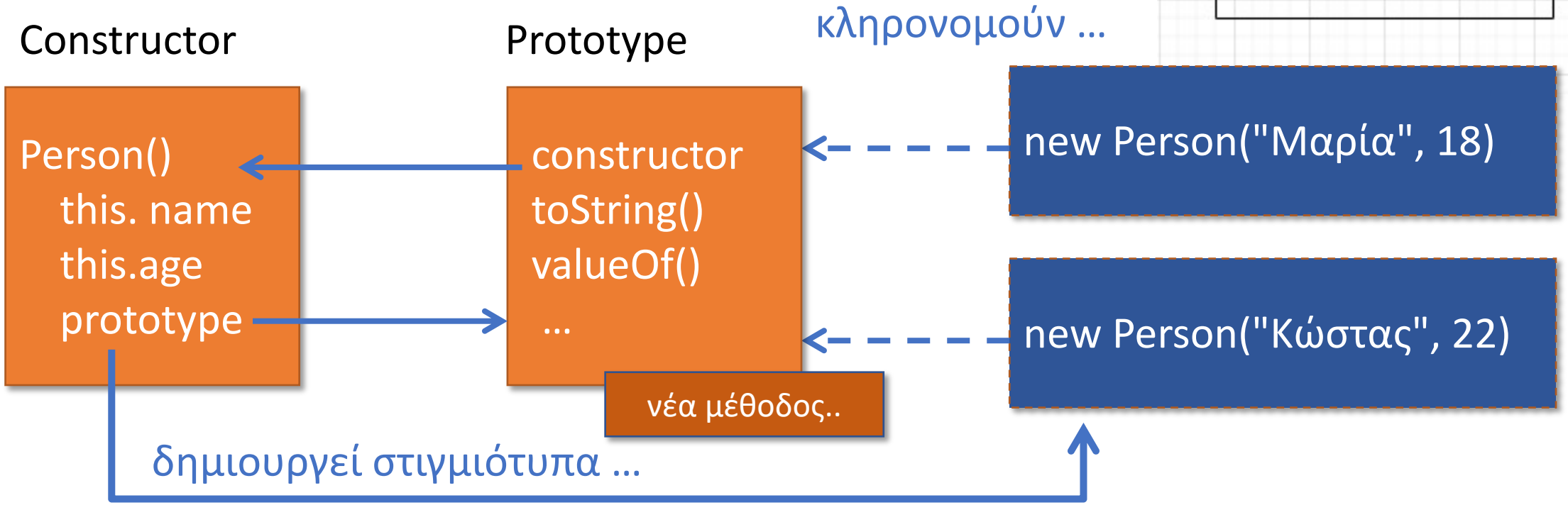
```
Car.prototype = {
  drive: function() {
    return `driving a ${this.make}...`;
  }
}

myCar.drive()
> 'driving a VW...'
```

ορισμός μεθόδου στην κλάση, μέσω του πρωτοτύπου της κλάσης

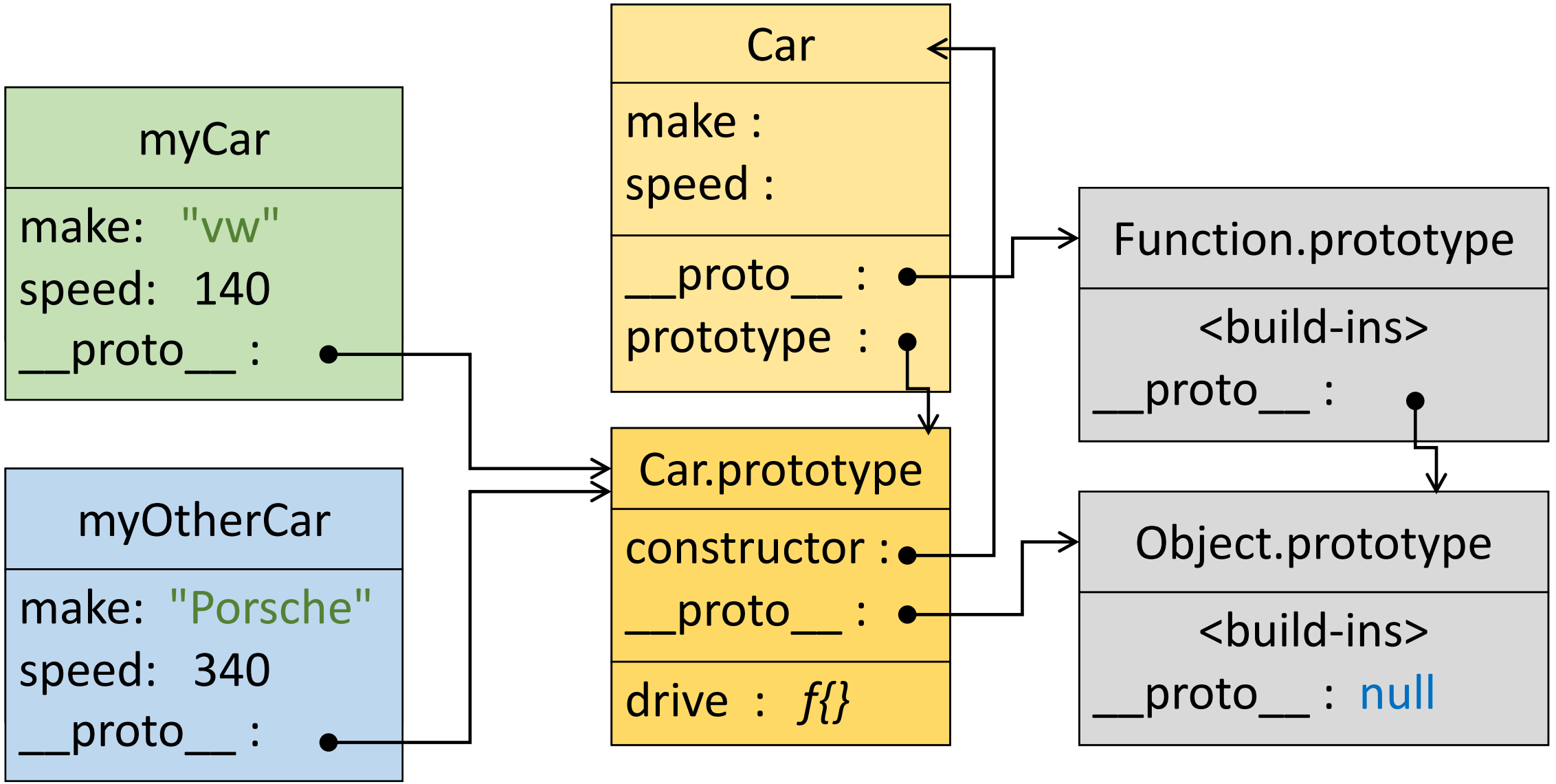
τι διαφορά έχει αν οριστεί η μέθοδος μέσα στον δημιουργό ή οριστεί στο πρωτότυπο;

δημιουργοί και πρωτότυπα





Αν προσθέσουμε μεθόδους ή ιδιότητες στο πρωτότυπο αυτές κληρονομούνται και από τα άλλα στιγμιότυπα της κλάσης


myCar
make: "vw"
speed: 140
__proto__ :
myOtherCar
make: "Porsche"
speed: 340
__proto__ :
Car
make :
speed :
__proto__ :
prototype :
Car.prototype
constructor :
__proto__ :
drive : f{}
Function.prototype
<build-ins>
__proto__ :
Object.prototype
<build-ins>
__proto__ : null

ιδιότητες πρωτοτύπου

κάθε στιγμιότυπο έχει τις εξής ιδιότητες και μεθόδους που κληρονομεί από το Object.prototype:

constructor – η συνάρτηση που κλήθηκε να δημιουργήσει το στιγμιότυπο, εδώ η συνάρτηση Object().

hasOwnProperty(ιδιότητα) - δέχεται συμβολοσειρά και δείχνει αν η ιδιότητα αυτή υπάρχει στο στγιμιότυπο. πχ. o.hasOwnProperty("name")).

isPrototypeOf(object) – δείχνει αν το αντικείμενο είναι πρωτότυπο ενός άλλου αντικειμένου

propertyIsEnumerable(propertyName) – δείχνει αν η ιδιότητα μπορεί να απαριθμηθεί πχ. με την εντολή for-in

toLocaleString() – Επιστρέφει την αναπαράσταση του αντικειμένου με βάση το τρέχον πλαίσιο

toString() – Επιστρέφει το αντικείμενο ως συμβολοσειρά

valueOf() - Επιστρέφει string, number, ή boolean ισοδύναμο, συνήθως επιστέφει το ίδιο με toString().

έλεγχος αν αντικείμενο ανήκει σε κλάση:

ο τελεστής `instanceof`

```
myCar instanceof Car;
> true
```

δημιουργία αντικειμένου με τον τελεστή class (ES6)

class syntax

```
class Person {
    constructor(name) {
        this.name = name; }
    greeting () {
            console.log('Γεια, είμαι ο ' + this.name);}
}
```

```
let p1 = new Person('Κώστας');
```

Η ιδιότητα __proto__

Τα αντικείμενα που δημιουργούνται με τους τρόπους που είδαμε περιέχουν το αντικείμενο **__proto__** που αφορά το **prototype**, μηχανισμό της JS για κληρονομιά ιδιοτήτων από το "object"

```
> p1
Person {name: "Κώστας"}
   name: "Κώστας"
   __proto__:
      constructor: class Person
      greeting: ƒ greeting()
      __proto__: Object
```

Κληρονομικότητα

```
class Player extends Person {
        greeting() {
                return super.greeting() + ', ο παίκτης'}
        }
```

```
const messi = new Player('Messi')
messi.greeting()
```

Επιστρέφει:

"Γεια σας είμαι ο Messi, ο παίκτης"

Συναρτήσεις get set

καλούνται όταν θέλουμε να δώσουμε τιμή ή να πάρουμε την τιμή μιας ιδιότητας

```
class Person {
        get fullName() {
                return `${this.firstName} ${this.lastName}` }
}

class Person {
        set age(years) {
                this.theAge = years}
}
```

παράδειγμα

```
class Person {
    constructor(name="", surname=""){
        this.name = name;
        this.surname = surname;
    }
    set fullName(x){
        [this.name, this.surname] = x.split(" ");
    }
    get fullName() {
        return this.name + " " + this.surname;
    }
}
```

```
let p = new Person();
p.fullName = 'Νίκος Καζαντζάκης';
p.fullName; // 'Νίκος Καζαντζάκης'
p.name; // 'Νίκος'
p.surname; // 'Καζαντζάκης '
```

Πρόσθετο υλικό

Διαχείριση συμβάντων (events)

Νίκος Αβούρης
Πανεπιστήμιο Πατρών

JavaScript: event based

Η εκτέλεση της JavaScript στον φυλλομετρητή ακολουθεί τη λογική των γλωσσών προγραμματισμού που στηρίζονται στο **μοντέλο χειρισμού συμβάντων (event-based model)**.
Τέτοιες είναι οι γλώσσες που λειτουργούν σε διαδραστικές συνθήκες, όπως σε γραφικά περιβάλλοντα αλληλεπίδρασης με τον χρήστη.

ακολουθία συμβάντων στο φόρτωμα ιστοσελίδας

Φάση 1. `document.readyState = "loading"`

Φάση 2. `document.readyState = "interactive"`

εκτέλεση κώδικα defer — event: `DOMContentLoaded`

Φάση 3. `document.readyState = "complete"`

event: `window.load`

Συμβάντα/ events

Events (συμβάντα) είναι **ενέργειες ή γεγονότα** που συμβαίνουν, για τα οποία το σύστημα μάς ενημερώνει, ώστε να μπορούμε να ανταποκριθούμε σε αυτά :

- Ολοκληρώθηκε η φόρτωση της ιστοσελίδας.
- Ο χρήστης κάνει κλικ με το ποντίκι πάνω από ένα συγκεκριμένο στοιχείο
- Ο χρήστης τοποθετεί τον δρομέα πάνω από ένα συγκεκριμένο στοιχείο.
- Ο χρήστης πατάει ένα πλήκτρο στο πληκτρολόγιο.
- Ο χρήστης αλλάζει μέγεθος ή κλείνει το παράθυρο του προγράμματος περιήγησης.
- Ένα βίντεο που αναπαράγεται ή τίθεται σε παύση ή τελειώνει.
- Παρουσιάστηκε κάποιο σφάλμα

Χειριστές συμβάντων / event handlers

Ένα συμβάν μπορεί να έχει ένα χειριστή συμβάντος **event handler**, που είναι ένα μπλοκ κώδικα (συνήθως μια συνάρτηση JavaScript) που θα τρέξει όταν προκύψει το συμβάν .

Όταν ορίζουμε το μπλοκ κώδικα να τρέχει ως απάντηση σε συμβάν αναφερόμαστε σε **καταχώρηση ενός χειριστή συμβάντος**.

Καταχώρηση χειριστή συμβάντων

1. Inline.

```
<button onclick="myfunc(this)">κλικ
</button>
```

2. Ιδιότητες αντικειμένων DOM on-event

```
window.onload = () => {
      //window loaded}
```

3. Με τον addEventListener()

```
window.addEventListener('load', () => {
      //window loaded
      })
```

1. inline χειριστής συμβάντων

Ένας τρόπος για ορισμό χειρισμού συμβάντων είναι να οριστεί ως γνώρισμα του στοιχείου HTML

```
<div onclick= "message()">όταν επιλεγεί αυτό το κείμενο θα κληθεί η συνάρτηση message</div>
```

Ο τρόπος αυτός δεν συνιστάται αφού έτσι ανακατεύουμε κώδικα HTML και Javascript

2. Ως ιδιότητα on-συμβάν του αντικειμένου του DOM

```
<button>Change color</button>
```

```
let btn = document.querySelector('button');

function rand (number) {
   return Math.floor(Math.random()*(number+1));
}
btn.onclick = function() {
   let rndCol = 'rgb(' + rand (255) + ',' + rand (255) + ',' + rand (255) + ')';
   document.body.style.backgroundColor = rndCol; }
```

Η ιδιότητα **onclick** είναι ιδιότητα του <button> αλλά είναι ένας ειδικός τύπος - όταν οριστεί ίσος με κάποιο κώδικα, ο κώδικας αυτός θα εκτελείται όταν το συμβάν προκύψει στο πλήκτρο

3. addEventListener() removeEventListener()

Οι addEventListener() and removeEventListener() χρησιμοποιούνται για δημιουργία και αναίρεση χειριστών συμβάντων.

Σύνταξη : `addEventListener(συμβάν, χειριστής)`

Πλεονέκτημα: ο event listener μπορεί να αναιρεθεί.

```
btn.addEventListener('click', bgChange);
```

συνάρτηση χειριστής συμβάντων

window.onload = f() ή
<html onload = f() >

Το <html onload= …> αφορά φόρτωμα του DOM,
το window.onload= … περιλαμβάνει επίσης φόρτωμα
των εικόνων και άλλων βοηθητικών αρχείων

```
<html onload = "function(e)"> // φορτώθηκε το DOM
```

```
window.onload = function(){
   …
} // Φορτώθηκε η ιστοσελίδα DOM και άλλα
βοηθητικά αρχεία
```

Άσκηση: περιγράψτε τη λειτουργία

```
<html>
<script>
    document.addEventListener('DOMContentLoaded', function(e) {
        alert("document loaded");})

    window.addEventListener('load', function(e) {
    console.log('window');
        alert("window loaded");})
</script>

<body>
    <p>my example</p>
    <img src="https://source.unsplash.com/featured/">
</body>
</html>
```

συνήθη συμβάντα

- element.onfocus και element.onblur —Συμβάντα που προκύπτουν όταν το btn έχει την εστίαση του δρομέα ή όχι
- element.ondblclick — Συμβάν που προκύπτει όταν κάνουμε διπλό κλικ στο btn.
- window.onkeydown, window.onkeyup — Ένα πλήκτρο πιέζεται στο πληκτρολόγιο. Το **keydown** και το **keyup** αφορούν το πλήκτρο κάτω και το πλήκτρο επάνω αντίστοιχα.
- element.onmouseover και element.onmouseout — Ο δείκτης μετακινείται ώστε να αιωρείται πάνω από το element, ή όταν απομακρύνεται από αυτό, αντίστοιχα.

ποιο το αποτέλεσμα;

```
<script>
   window.onload = function() {
      p = document.querySelector("myp");
      i = 0;
      while (true) {
         p.innerHTML += i + " * ";
         i++;
         if (i == 10) break;
      }
   }
</script>
```

21. Διαχείριση συμβάντων: παράδειγμα

Νίκος Αβούρης
Πανεπιστήμιο Πατρών

onmousedown onmouseup

Πατήστε στη λάμπα!

Άσκηση: αλλάξτε τον κώδικα ώστε να ανάβει/ σβήνει όταν πατάμε στη λάμπα

```
<html lang='el'>
<head>
<style>
        body {
                font-family: sans-serif;
        }
</style>
<script>
        function lighton() {
                document.querySelector('myimage').src = "bulbon.gif";
        }
        function lightoff() {
                document.querySelector('myimage').src = "bulboff.gif";
        }
</script>
</head>
<body>
        <img id="myimage" onmousedown="lighton()"
onmouseup="lightoff()" src="bulboff.gif" width="100" height="180" />
        <p>Πατήστε στη λάμπα!</p>
        <p>Άσκηση: αλλάξτε τον κώδικα ώστε να ανάβει/ σβήνει
        <br>όταν πατάμε στη λάμπα</p>
</body>
</html>
```

global object

(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects)

Τα στοιχεία στο global object της JavaScript

Συναρτήσεις

* **eval(έκφραση)** : υπολογισμός του αποτελέσματος της έκφρασης,
* **isFinite(εκφραση)** : επιστρέφει `false` αν το αποτέλεσμα της έκφρασης είναι +Infinity, –Infinity, NaN ή undefined, αλλιώς `true`
* **isNaN(τιμή)** : επιστρέφει `true` αν η τιμή επιστρέφει NaN, αλλιώς `false`.
* **parseFloat(συμβολοσειρά)** : επιστρέφει την αριθμητική τιμή ως δεκαδικό αριθμό, αν η συμβολοσειρά αρχίζει με μη αριθμητικό χαρακτήρα επιστρέφει NaN
* **parseInt(συμβολοσειρά, βάση)** : παρόμοια με την parseFloat() για ακέραιο αριθμό, παίρνει ως δεύτερο όρισμα τη βάση του αριθμητικού συστήματος (πχ 16 για το δεκαεξαδικό, κλπ.)
* **encodeURI(URI)** : Κωδικοποίηση δεδομένων με την τεχνική URL Encoding, βάσει της κωδικοσελίδας UTF-8, για παράδειγμα κωδικοποιεί τα κενά ως %20.
* **encodeURIComponent()** : όπως η προηγούμενη μόνο που αφορά δεδομένα που τα τοποθετηθούν σε μεταβλητές PUT GET
* **decodeURI()** : αποκωδικοποίηση της encodeURI()
* **decodeURIComponent()** : αποκωδικοποίήση της encodeURIComponent()

Τα στοιχεία στο global object της JavaScript

Βασικά Αντικείμενα

* **Object** : το αρχέτυπο αντικείμενο, τα περισσότερα αντικείμενα της JavaScript κληρονομούν από αυτό.
* **Function** : η κλάση των συναρτήσεων της JavaScript που είναι αντικείμενα πρώτης τάξης.
* **Boolean** : αντικείμενο που αντιστοιχεί σε λογικές τιμές
* **Symbol** : αντικείμενο που αντιστοιχεί στα δεδομένα τύπου Symbol

Αντικείμενα Αριθμών

* **Number** : αντικείμενο που αντιστοιχεί στον πρωτογενή τύπο δεδομένων Number
* **BigInt** : αντικείμενο που αντιστοιχεί στον πρωτογενή τύπο δεδομένων BigInt για άριθμους μεγαλύτερους από 2⁵³ – 1
* **Math** : αντικείμενο με ιδιότητες και μεθόδους για μαθηματικές συναρτήσεις και σταθερές.
* **Date** : αντικείμενο που εκφράζει χρονική στιγμή σε milliseconds με αφετηρία μέτρησης 1 Ιανουαρίου 1970 UTC (τύπου Number)

Τα στοιχεία στο global object της JavaScript

Κείμενο

* **String** : αντικείμενο που αφορά την αναπαράσταση και μεθόδους που αφορούν συμβολοσειρές
* **RegExp** : αντικείμενο που επιτρέπει την αντιστοίχιση προτύπων χαρακτήρων (κανονικές εκφράσεις) με συμβολοσειρές

Συλλογές αντικειμένων

* **Array** : αντικείμενο που αφορά την κλάση αντικειμένων τύπου Array (πίνακες), να σημειωθεί ότι υπάρχουν ακόμη αντικείμενα για typed Arrays, πίνακες που δέχονται ορισμένου τύπου δεδομένα, όπως Float32Array, κλπ.
* **Map** : αντικείμενο που αφορά την κλάση Map, ακολουθία ζευγών κλειδί: τιμή που θυμάται τη σειρά δημιουργίας της.
* **Set** : αντικείμενο που αφορά την κλάση Set, που επιτρέπει την αποθήκευση συλλογής μοναδικών στοιχείων

Σφάλματα

* **Error** : αντικείμενα τα οποία δημιουργούνται όταν συμβούν σφάλματα κατά τη διάρκεια εκτέλεσης του κώδικα, υπάρχουν ειδικοί τύποι σφαλμάτων, πχ ReferenceError, κλπ.

αντικείμενα του global scope

Συναρτήσεις αριθμών

`int` **`parseInt( string data [, int radix ] )`**	Εξαγωγή ενός ακέραιου αριθμού από συμβολοσειρά. Το προαιρετικό όρισμα εισόδου radix επιτρέπει τη σωστή αναγνώριση του αριθμού σε ένα από τα αριθμητικά συστήματα: δεκαδικό (10), δεκαεξαδικό (16), δυαδικό (2) ή οκταδικό (8).
`float` **`parseFloat( string data )`**	Εξαγωγή ενός πραγματικού αριθμού από συμβολοσειρά.
`bool` **`isNaN( mixed numdata )`**	Έλεγχος αν η μεταβλητή είναι (ακέραιος ή πραγματικός) αριθμός ή συμβολοσειρά που αναπαριστά αριθμό.
`bool` **`isFinite( mixed numdata )`**	Έλεγχος αν η μεταβλητή είναι έγκυρος, πεπερασμένος (Finite) ακέραιος ή πραγματικός αριθμός, ή συμβολοσειρά που αναπαριστά πεπερασμένο αριθμό.

Συναρτήσεις κωδικοποίησης-αποκωδικοποίησης URL

`string` **`encodeURI( string uri )`** `string` **`encodeURIComponent( string uriQuery )`**	Κωδικοποίηση δεδομένων βάσει της τεχνικής URL Encoding βάσει της κωδικοσελίδας UTF-8. Η πρώτη χρησιμοποιείται για την κωδικοποίηση ενός πλήρους URI ενώ η δεύτερη μόνο για καθαρά δεδομένα που θα τοποθετηθούν σε GET μεταβλητές.
`string` **`decodeURI( string uri )`** `string` **`decodeURIComponent( string uriQuery )`**	Αποκωδικοποίηση δεδομένων από URL Encoding με κωδικοσελίδα UTF-8.

Το αντικείμενο Math

Νίκος Αβούρης
Πανεπιστήμιο Πατρών

μέθοδοι του Math

```
Math.abs(x), Math.ceil(x), Math.cos(x)
Math.exp(x), Math.floor(x), Math.log(x),
Math.max(a,b), Math.min(a,b),
Math.pow(x,y), Math.round(x), Math.sin(x),
Math.sqrt(x), Math.tan(x) Math.PI, Math.E
```

Άσκηση

Να κατασκευάσετε συνάρτηση που επιστρέφει το μέγιστο 3 αριθμών Σημείωση: χρησιμοποιήστε τη μέθοδο: Math.max(x,y)

```
function maximum( x, y, z ) {
        return Math.max( x, Math.max( y, z ) );
        }
```

Math.random()

Να κατασκευάσετε μια γεννήτρια τυχαίων αριθμών που προσομοιώνει το ρίξιμο του ζαριού 20 φορές

```
<script>
let table = document.querySelector("dice");
let cell;
let value;
let row = document.createElement('tr');
for ( let i = 1; i <= 20; i++ ) {
    value = Math.floor( 1 + Math.random() * 6 );
    cell = document.createElement('td');
    cell.textContent = value;
    row.appendChild(cell);
    if ( i % 5 === 0 ){
        table.appendChild(row);
        if (i !== 20)
        row = document.createElement('tr');
        } // end if
} // end for
</script>
```

Το αντικείμενο Date

το αντικείμενο Date

```
let newDate = new Date();
```

Sat Feb 02 2030 12:03:48 GMT+0200
(Eastern European Standard Time)

το αντικείμενο Date

getFullYear()	Ανάκτηση του **έτους** ως τετραψήφιου αριθμού (yyyy)
getMonth()	Ανάκτηση του **μήνα** ως αριθμού (0-11)
getDate()	Ανάκτηση της **ημέρας** ως αριθμού (1-31)
getHours()	Ανάκτηση της **ώρας** (0-23)
getMinutes()	Ανάκτηση των **λεπτών** (0-59)
getSeconds()	Ανάκτηση **δευτερόλεπτων** (0-59)
getMilliseconds()	Ανάκτηση **millisecond** (0-999)
getTime()	Ανάκτηση του χρόνου (milliseconds από 1, Ιανουαρίου 1970)
getDay()	Ανάκτηση της ημέρας ως αριθμού (0-6), 0=Sun, 1=Mon, …
Date.now()	Ανάκτηση του χρόνου. ECMAScript 5.

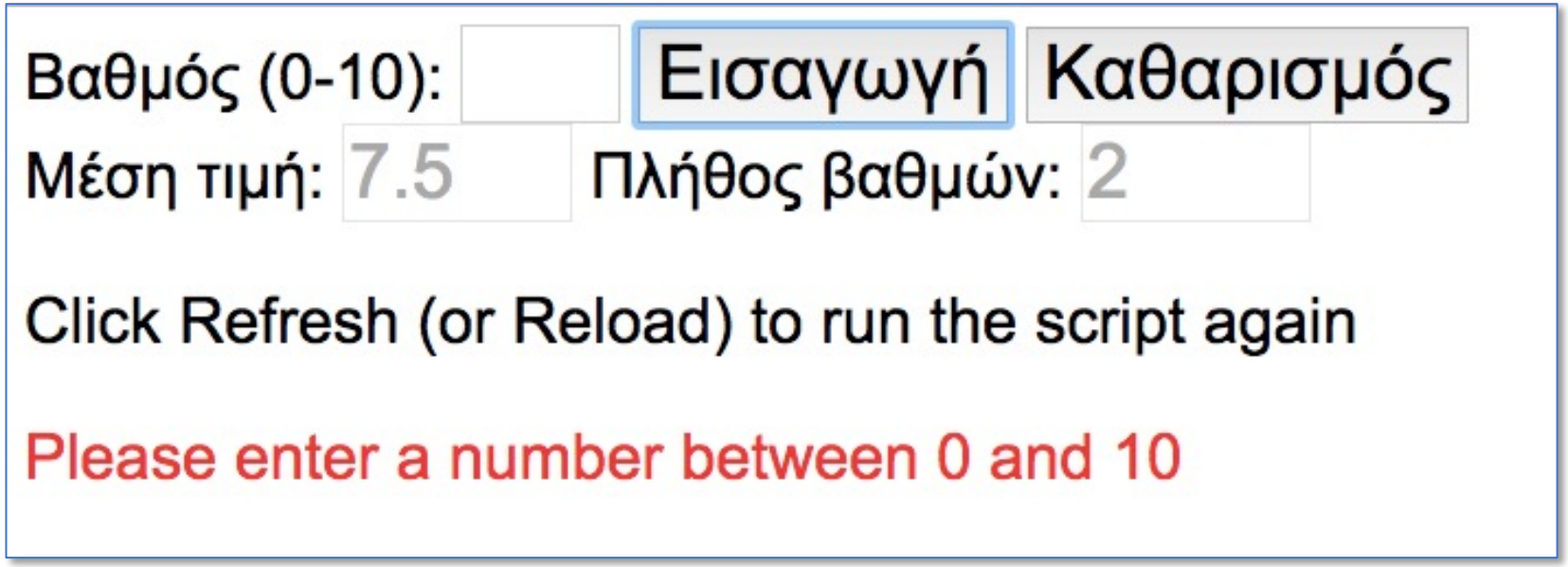
object **Date**

```
let dateString = "Σήμερα είναι: ";
let newDate = new Date(); // Get the month, day,
and year.
dateString += newDate.getDate() + "/"; dateString
+= (newDate.getMonth() + 1) + "/";
dateString += newDate.getFullYear();
console.log(dateString);
```

Άσκηση: φόρμα βαθμολογίας

Άσκηση

Να δημιουργήσετε μια φόρμα που επιτρέπει την εισαγωγή βαθμών στην κλίμακα 0-10 και υπολογίζει τη μέση τιμή και το πλήθος των βαθμών.

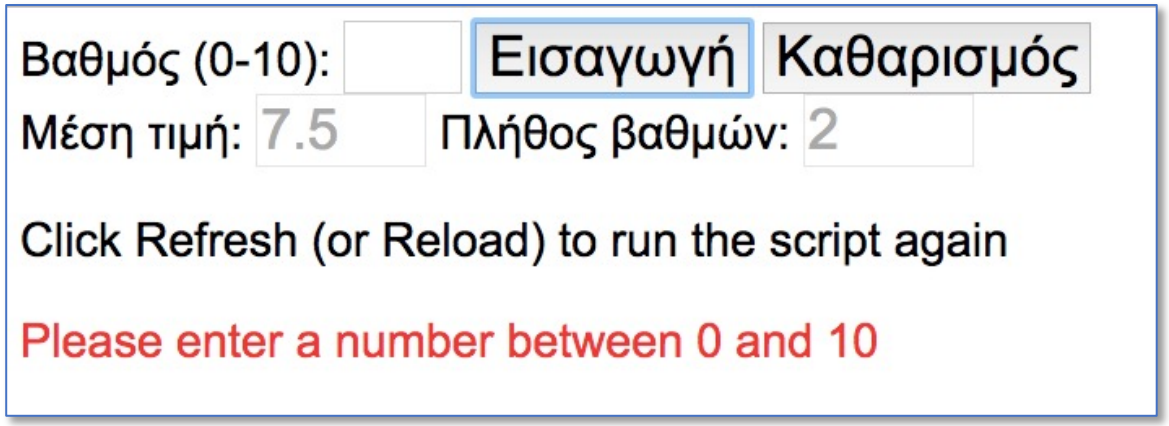


ex42-grades.html

Άσκηση: διόρθωση της φόρμας

Άσκηση

Να διορθώσετε τη φόρμα του προηγούμενου παραδείγματος ώστε:
(α) Να εισάγεται ο βαθμός με <Enter> χωρίς αναγκαστικά να πατάει ο χρήστης το πλήκτρο "Εισαγωγή"
(β) Να γίνει διαχωρισμός του κώδικα JS από την HTML



ex42-grades.html

```
const newVathmos = document.querySelector("vathm");


 newVathmos.addEventListener("keyup", (e) =>
                                {if (e.code === "Enter"){
                                          calculate();
                                }});
```