ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Διαδικτυακά Πολυμέσα και Γραφικά

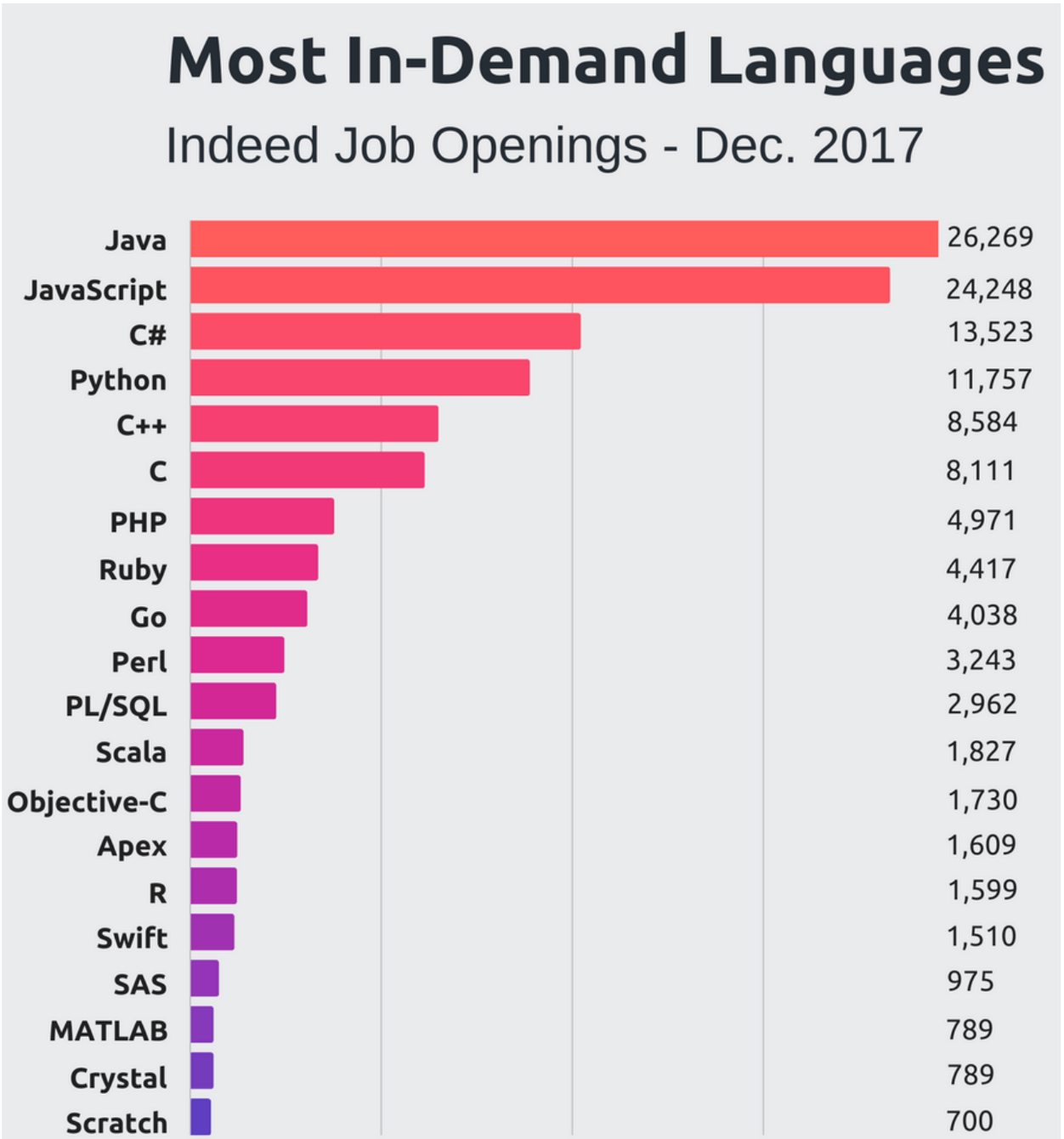
Javascript

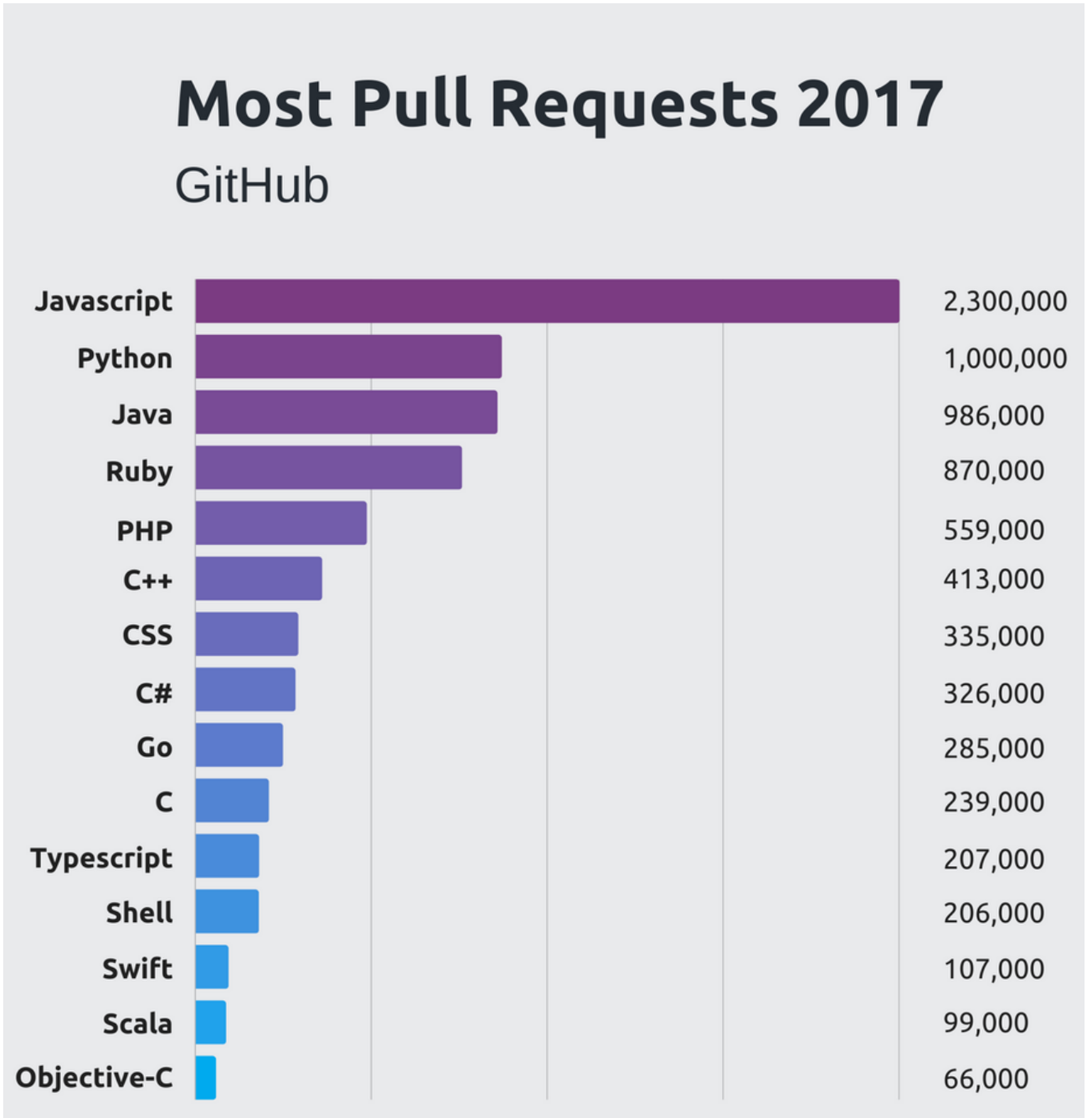
Στέλιος Σφακιανάκης

Χειμερινό 2021

Τί είναι;

- Γλώσσα Προγραμματισμού για τη δημιουργία “**δυναμικών**” σελίδων στον Παγκόσμιο Ιστό.
  - “*Δυναμική*”, “*διερμηνευόμενη*” (interpreted), με “*ασθενείς τύπους*” (weakly typed)
- Η πιο δημοφιλής και ευρέως χρησιμοποιούμενη γλώσσα προγραμματισμού
  - Καθε πλοηγητής ιστού (web browser) διαθέτει μια μηχανή εκτέλεσης Javascript
  - “Ακόμα και η γιαγιά σας την χρησιμοποιεί!”
- Δημιουργήθηκε (μέσα σε 10 μέρες!) το 1995 από τον **Brendan Eich** του Netscape Communications
- *Δεν έχει καμία σχέση με την Java!!* Ονομάστηκε έτσι για εμπορικούς λόγους





https://stackify.com/popular-programming-languages-2018/

Η Javascript ως γλώσσα προγραμματισμού*

* Με έμφαση στην μοντέρνα και στάνταρτ έκδοση της, "ECMAscript 2015" (ES-6)
Reference and Tutorials: https://developer.mozilla.org/en-US/docs/Web/JavaScript
Επίσης : http://speakingjs.com/es5/

Πριν προχωρήσουμε: Javascript vs ECMAScript

- Όλοι τη λέμε JavaScript, αλλά το όνομα της ειναι trademark της Oracle (που το "κληρονόμησε" από τη Sun)

- Οπότε, το official όνομα της ειναι **ECMAScript** που προτυποποιείται από τον οργανισμό ECMA

- Υπαρχουν διαφορετικές "εκδόσεις" της προδιαγραφής (specification) της ECMAScript

  - Από τις πρόσφατες εκδόσεις, περισσότερο ευρέως υποστηριζόμενη ειναι η 6η εκδοση ECMAScript 2015 ή ECMAScript 6

  - Υποστήριξη από πλοηγητές : http://kangax.github.io/compat-table/es2016plus/

Βασικό συντακτικό

```
// Σχόλιο μονής γραμμής

/*
 Σχόλιο που μπορεί να περιέχει
 περισσότερες από
 μια γραμμές
*/
var x;  // Ορισμός μεταβλητής

x = 3 + y;  // ανάθεση τιμής σε μια μεταβητη `x`


foo(x, y);  // κλήση συνάρτησης `foo` με παραμέτρους `x` και `y`
obj.bar(3);  // κλήση μεθόδου `bar` του αντικειμένου `obj`

// υπο-συνθήκη έλεγχος και εκτέλεση ενος μπλοκ εντολών
if (x === 0) {  // Εαν το `x` είναι ίσο με το μηδέν...
    x = 123;
}

// Δήλωση συνάρτησης `baz` με παραμέτρους `a` και `b`
function baz(a, b) {
    return a + b;
}
```

Μεταβλητές και Εντολές Εκχώρησης τιμής

- Όπως και σε άλλες γλώσσες προγ/μου οι μεταβλητές είναι "αποθήκες τιμών" (ονόματα θέσεων μνήμης που περιέχουν τιμές)

- Τα ονόματα των μεταβλητών στην Javascript αποτελούνται από Unicode χαρακτήρες: γράμματα, αριθμοί, και σύμβολα (π.χ. `$`, `_`) και είναι case-sensitive

    - Παραδείγματα: `var1`, `$myvar`, `_tst_`, ...

    - Κάποια ονόματα (π.χ. `true`, `false`, `class`, `try`, `catch`, `this`, ...) είναι δεσμευμένα και δεν μπορούν να χρησιμοποιηθούν ως ονόματα μεταβλητών

- Αλλάζουμε την τιμή μιας μεταβλητής με τον τελεστή εκχώρησης τιμής (`=`)

- Σε αντίθεση με την C αλλά όπως και π.χ. στην Python, δεν δηλώνουμε τον τύπο δεδομένων μιας μεταβλητής (δηλ. αν είναι π.χ. αριθμός ή string)

Δήλωση Μεταβλητών

- Οι μεταβλητές εισαγονται/δηλώνονται με τα `var`, `let`, και `const` (χωρίς να δηλώνεται ο τύπος δεδομένων):
  - `const`, για σταθερές τιμές (π.χ. το π = 3,14..)
  - `let`, για "κανονικές" μεταβλητές

ES6

- Σε αντίθεση με το `var`, τα `let` και `const` είναι *scope* (block) aware οπότε και προτιμητέα
  - Το scope είναι εντολές που είναι διαχωρίζονται από τον υπόλοιπο κώδικα με αγκύλες

```
const name = "Stelios";
let age = 43;
age++; // 44
name = "Manos";// TypeError: Attempted to assign to readonly property.

if (true) {
   const name = "George";
   console.log(name); // George
}
console.log(name); // Stelios
age = "Foo" // No error, στο εξής το age θα είναι τύπου string με τιμή 'Foo'
```

Βασικοί Τύποι Δεδομένων

- Αριθμοί (numbers)
- Λογικοί (boolean)
- Αλφαριθμητικά (strings)
- Ειδικοί τύποι με αντίστοιχες τιμές: `null`, `undefined`

Πρωταρχικοί (primitive) τύποι

- Αντικείμενα (objects)

Αριθμοί

- **Number**: Κοινός τύπος για ακεραίους και αριθμούς κινητής υποδιαστολής! Δηλ. όλοι οι αριθμοί στην Javascript αναπαρίστανται εσωτερικά ώς doubles floating numbers.

- Ευρος τιμών μεταξύ $-(2^{53} - 1)$ και $2^{53} - 1$ (IEEE-754 64bits)

- Ειδικές τιμές : `+Infinity`, `-Infinity`, and `NaN` (not-a-number)
    - Π.χ. `1/0 == Infinity`,

- Δυαδική αναπαράσταση: `0b1011`, 16αδική: `0xBADCAFE3`

- Εκθετική αναπαράσταση : `2e10`, `6.626068E-34`

Αριθμητικοί τελεστές και μέθοδοι

- Οι συνήθεις τελεστές:
  - Αριθμητικοί: `+` , `-` , `*` , `/` , `%` , `**` (ύψωση σε δύναμη), και οι μοναδιαίοι `++` (αύξηση κατα ένα), `--` (μείωση κατά ένα)
  - Αριθμητικοί χειρισμού bits (not, and, or, xor, ..): `~`, `&`, `|`, `^`, `>>`, `<<`, `>>>`
- Το `Math` object παρέχει συνήθεις μαθηματικές συναρτήσεις, όπως `log`, `log10`, `exp`, `sqrt`, `abs`, τριγωνομετρικές (`cos`, `sin`, `tan`, ...), στρογγυλοποίησης (`round`, `floor`, `ceil`), κλπ
  - Πλήρης τεκμηρίωση στο http://mdn.io/math

Booleans

- **Boolean** με δύο τιμές: `true` και `false`.
- Γενικα, μη αληθείς τιμές θεωρούνται τα `false`, `0`, `""`, `null`, `undefined`, και `NaN`
- Τελεστές και πράξεις:
  - Οι συνήθεις λογικοί τελεστές: `&&` (and) `||` (or) `!` (not)
  - Τελεστές σύγκρισης: `<` , `>` , `>=` , `<=` , `==` , και οι τελεστές "αυστηρής" ισότητας: `===`, `!==`

Strings

- **Strings**: αλφαριθμητικά (πινακες από χαρακτήρες) με υποστήριξη Unicode, π.χ. `"Stelios"`, `"أنا بحبك"`, `'Maria 2018'`
  - Χρησιμοποιούνται μονά (`'`), διπλά (`"`), ή “ανάποδα μονά” (`` ` ``) εισαγωγικά
- Ως πίνακες απο χαρακτήρες, μπορούμε να προσπελάσουμε τον χαρακτήρα βασει της θέσης του (index), ξεκινώντας από το 0. Πχ. `"Stelios"[2] === 'e'`
- "Ενωση" (concatenation) δύο αλφαριθμητικών μέσω του τελεστή "+" (πρόσθεση):
  - `"Hello " + "World"` μας δίνει το `"Hello World"`
- Χρήσιμες μέθοδοι: `indexOf()`, `charAt()`, `match()`, `search()`, `replace()`, `toUpperCase()`, `toLowerCase()`, `slice()`, `substr()`, ...

String functions

`s.length` : επιστρέφει το μήκος του `s`

`s.charAt(k)` : επιστρέφει το `s[k]`. Το `k` ξεκινάει από το 0

`s.charCodeAt(k)` : Η Unicode τιμή του `s[k]`

`s1.indexOf(s2,k)` : επιστρέφει τη θέση εμφάνισης το `s2` στο `s1` ξεκινώντας από τη θέση `k`. Αν δεν δοθεί το k τότε θεωρείται το 0 η θέση εκκίνησης. Αν δεν βρεθεί επιστρέφει -1

`s1.lastIndexOf(s2)` : επιστρέφει την τελευταία θέση εμφάνισης του `s2` στο `s1`. Αν δεν βρεθεί επιστρέφει -1

`s1.replace(s2,s3)`: αντικαθιστα μεσα στο `s1` το `s2` με το `s3`

`s.slice(i,j)` : επιστρέφει το substring που ξεκινάει από τη θέση `i` μέχρι και το χαρακτήρα που προηγείται της θέσης `j`. Αν δεν δοθεί το `j` τότε επιστρέφει μέχρι το τέλος. Αν `i < 0` τότε επιστρέφει από το τέλος.

`s.substr(k,len)` : επιστρέφει το substring που ξεκινάει από τη θέση `k` και έχει `len` χαρακτήρες. Αν δεν δοθεί το len τότε παίρνει μέχρι το τέλος. Αν το `k` είναι αρνητικό τότε ξεκινάει από τη θέση `s.length + k`

`s.toLowerCase()` : επιστρέφει το `s` με "μικρά" γράμματα

`s.toUpperCase()` : επιστρέφει το `s` με κεφαλαία γράμματα

ES6 Template strings

- Επιτρέπουν την ενσωμάτωση μεταβλητών και εκφράσεων μεσα σε αλφαριθμητικά

- Επιτρέπουν αλφαριθμητικά πολλαπλών γραμμών

- Περικλείονται με το backtick (`)

```
let name = "George";
let age = 43;
let message = `Hi ${name}, next year you will be ${age+1} years old`;
```

Αντικείμενα

sos!

- Η Javascript είναι αντικειμενοστραφής!
- Όλα τα non-primitive values είναι objects (δηλ. ό,τι δεν είναι αριθμός, boolean, κλπ)
- Τα αντικείμενα είναι σύνθετοι τύποι, και μπορεί να είναι:
  - Απλά αντικείμενα, που τα εισάγουμε με αγκύλες (`{..}`)
  - Ημερομηνίες (`Date`)
  - Κανονικές εκφράσεις (`RegExp`)
  - `Set`, `Map` (νέα στην EcmaScript 6): Σύνολα και "Λεξικά"
  - Ή ακόμα αντικείμενα που ορίζουμε εμείς!

Αντικείμενα

- Τα αντικείμενα είναι mappings (*αντιστοιχήσεις*) μεταξύ "κλειδιών" (*keys*, *properties*) και τιμών
  - Τα κλειδιά είναι strings ενώ οι τιμές μπορεί να είναι οτιδήποτε
  - Τα κλειδιά δεν είναι απαραίτητο να έχουν εισαγωγικά εκτός ..αν είναι απαραίτητο :-) (π.χ. αν περιέχει το κενό)
- Μπορούμε να τα δημιουργήσουμε με το "`new Object()`" ή (συνηθέστερα) ως "*object literal*" με αγκύλες που περιέχουν τα properties χωρισμένα με κόμμα.
- Μπορούμε να διαβάσουμε (get) ή να αλλάξουμε (set) την τιμή ενός property σε ένα αντικείμενο

```
let person = {name: "Maria",
              "e-mail": "mar@d.com",
               age: 25};

person["e-mail"];  // "mar@d.com"

person.name; // "Maria"
person.age; // 25

delete person.age;

Object.keys(person); // [ 'name', 'e-mail' ]
```

- Αφαίρεση ενός property με το `delete`, και απαρίθμηση τους με το `Object.keys()`
  - Προσοχή: θέτοντας ως τιμή του property το `null` δεν διαγράφει το property από το αντικείμενο
  - Άλλες συναρτήσεις: `Object.values()` για να πάρουμε όλες τις τιμές σε ένα array, `Object.entries()` για να πάρουμε τα ζευγάρια properties και τιμών τους σε array
- Μπορούμε να χρησιμοποιήσουμε τον τελεστή `in` για να ελέγξουμε αν ένα αντικείμενο περιέχει ένα property και να χειριστούμε επαναληπτικά τα properties με την δομή `for/in`.
- Η συνάρτηση `Object.assign()` αντιγράφει όλα τα properties ενός αντικειμένου σε ένα άλλο

```
let person = {name: "Maria",
              age: 25};

Object.keys(person);    // [ 'name', 'age' ]
Object.values(person);  // [ 'Maria', 25 ]
Object.entries(person); // [ ['name', 'Maria'], ['age', 25] ]

'name' in person;    // true
```

```
for (let p in person)
   console.log(p + "=" + person[p]);    // name=Maria
                                        // age=25
person.sayHi = function() { console.log(`Hi, this is ${this.name}`); }
person.sayHi() // Hi, this is Maria

let another = {name: 'John'};
Object.assign(another, person);
another.name; // 'Maria'
another.age;  // 25
```

Πίνακες (Arrays)

- **Arrays** (πίνακες): αριθμημένες λίστες από "πράγματα", σε σειρά ξεκινώντας από το 0

  - `arr = [...]` ή `arr = new Array(...)`

- Για να προσπελάσουμε ένα στοιχείο χρησιμοποιούμε τη θέση του (index): `arr[1]` -> 2ο στοιχείο στο array

- Το `.length` δίνει το πλήθος των στοιχείων του array

Μέθοδοι των Arrays

- `arr.join(str)`: "ενώνει" όλα τα στοιχεία του array arr σε ενα string χωρισμένα με το string `str`

- `arr1.concat(arr2)`: επιστρέφει ενα νέο array που περιέχει όλα τα στοιχεία των `arr1` και `arr2`

- `arr.slice(j, k)`: επιστρέφει ενα νέο array που περιέχει όλα τα στοιχεία του `arr` από τη θέση `j` μέχρι και τη θέση πριν το `k` (δηλ. το `k-1`). Αν λείπει το `k` τότε παίρνει μεχρι το τέλος, ενώ αν δεν υπάρχει ούτε το `j` τότε επιστρέφει ένα αντίγραφο όλου του array.

- `arr.sort()`: ταξινομεί το array "in place" (δηλ. το τροποποιεί)

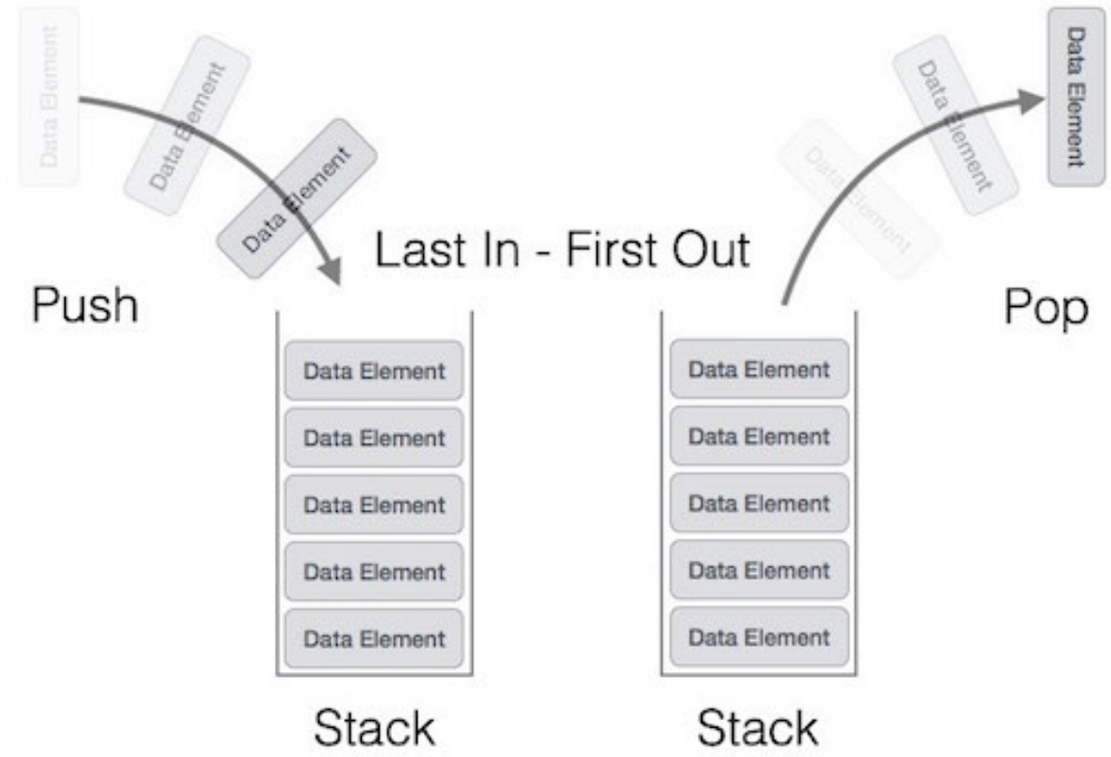
Άλλες μέθοδοι

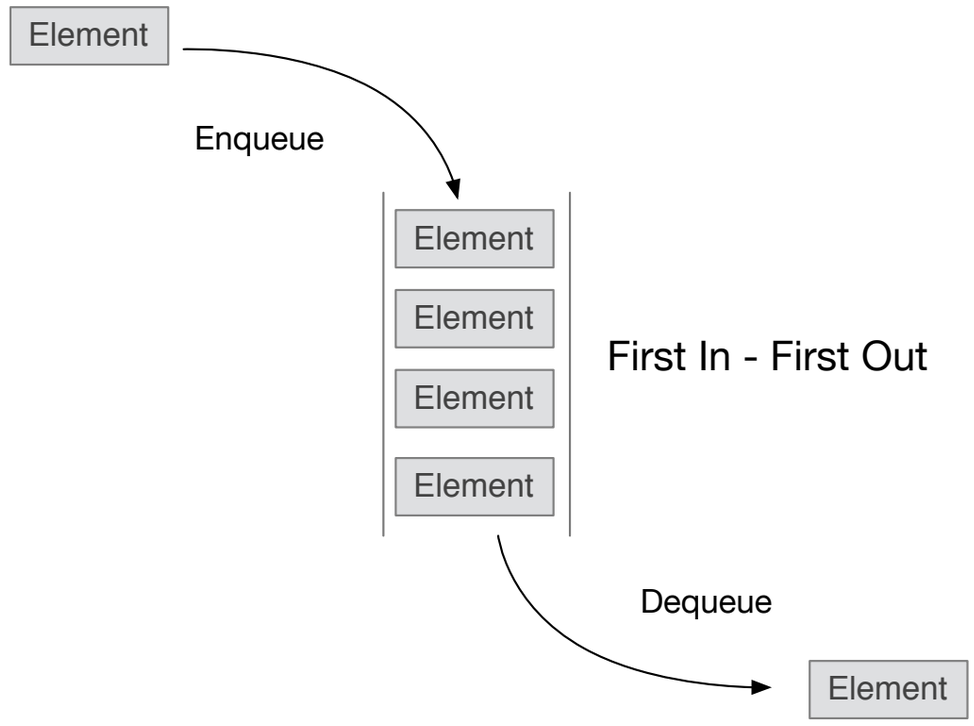
- Μέθοδοι που δουλεύουν στο "τέλος" του array:
  - `arr.push(elem)` : εισαγωγή στοιχείου `elem` στο τέλος του `arr`
  - `arr.pop()` : εξαγωγή του τελευταίου στοιχείου
- Μέθοδοι που δουλεύουν στην αρχή του array:
  - `arr.shift()` : εξαγωγή του πρώτου στοιχείου
  - `arr.unshift(elem)` : εισαγωγή στοιχείου `elem` στην αρχή του `arr`

Πίνακες (Arrays)

```
fruits = ["Apple", "Orange", "Plum"];

alert( fruits.length );  // 3

alert( fruits[0] );      // Apple
alert( fruits[1] );      // Orange
alert( fruits[2] );      // Plum

fruits[2] = 'Pear';    // τώρα ["Apple", "Orange", "Pear"]
fruits.push('Melon'); // τώρα ["Apple", "Orange", "Pear", "Melon"]

fruits.shift();              // τώρα ["Orange", "Pear", "Melon"]
fruits.unshift('Avocando'); // τώρα ["Avocando","Orange", "Pear", "Melon"]
```

Παράδειγμα: Υλοποίηση "στοίβας" και "ουράς"





- Με ποιο συνδυασμό μεθόδων Arrays μπορούμε να υλοποιήσουμε μια στοίβα ή μια ουρά;

Δομές ελέγχου

Δομές Ελέγχου - Conditionals

- Υπό Συνθήκη (Conditionals): `if/else` και `switch/case`
  - Όπως και στη C, C++, Java, ...
- Στην if χρησιμοποιούμε μια λογική έκφραση για να επιλέξουμε ποιό από τα δύο θα κάνουμε (αν είναι `true` ή `false`)
  - Το τί θα κανουμε μπορεί να είναι περισσότερες απο μία εντολές οπότε περικλείονται μέσα σε αγκύλες
- Στην switch ελέγχεται η τιμή μιας έφρασης αν είναι ίση με καποια από ένα σύνολο τιμών
- Υπάρχει ακόμα και ο τριαδικός τελεστής "`<condition> ? <then> : <else>`" για το υπολογισμό μιας υπο συνθήκη έκφρασης

```
if (myvar === 0) {
    // τότε
}

if (myvar === 0) {
    // τότε
} else {
    // αλλιώς
}

if (myvar === 0) {
    // τότε
} else if (myvar === 1) {
    // αλλιώς αν
} else if (myvar === 2) {
    // αλλιως-αν
} else {
    // αλλιώς
}
```

```
switch (fruit) {
    case 'banana':
        // ...
        break;
    case 'apple':
        // ...
        break;
    default:  // όλες οι άλλες περιπτώσεις
        // ...
}
```

Παράδειγμα: Δίσεκτο Έτος

- Για να είναι δίσεκτο ένα έτος πρέπει να διαιρείται ακριβώς με το 4. Αν όμως είναι επαιώνιο έτος (αν είναι δηλαδή έτος που ξεκινά έναν αιώνα, π.χ. 1900, 2000, 2100, κλπ.) πρέπει να διαιρείται και με το 400.

- Π.χ. δίσεκτα είναι τα : 2000, 2016, 2022

- Ενώ δεν είναι τα 1900, 2001, 2100

```
var is_leap_year;
if (year % 100 == 0) {
    if (year % 400 == 0)
        is_leap_year = true;
    else
        is_leap_year = false;
}
else {
    if (year % 4 == 0)
        is_leap_year = true;
    else
        is_leap_year = false;
}
```

```
is_leap_year = year % 100 == 0 ? year % 400 == 0 : year % 4 == 0;
```

Ισότητες: Προσοχή!! (1)

```
1    ==  1          // true
'1'  ==  1          // true
1    == '1'         // true
0    == false       // true
0    == null        // false
0    == undefined   // false
null == undefined   // true
```

- Οι τελεστέοι μετατρέπονται αν δεν έχουν τον ίδιο τύπο (π.χ. αριθμός με string) και μετά γίνεται ο έλεγχος της ισότητας

- ***Συμβουλή: χρησιμοποιούμε πάντα το*** `===` ***(και το*** `!==`***) που δεν κάνει μετατροπή των τελεστέων: επιστρέφει πάντα*** `false` ***(αντίστοιχα*** `true`***) αν εχουμε διαφορετικούς τύπους!**

- Γενικά αποφεύγουμε να αναμειγνύουμε διαφορετικούς τύπους! Δείτε το https://www.destroyallsoftware.com/talks/wat (από το 1:21 και μετά)

Ισότητες: Προσοχή!! (2)

- Επίσης προσοχή : αν συγκρίνουμε αντικείμενα η ισότητα θα είναι αληθής μόνο όταν οι δύο "τελεστέοι" αναφέρονται στο ίδιο αντικείμενο

- Στην περίπτωση των πρωταρχικών τύπων (αριθμοί, strings) δεν υπάρχει τετοιο πρόβλημα

```
var a = {};
var b = {};

a === b; // false!

// όμως αν οι μεταβλητές αναφέρονται
// στο ίδιο αντικείμενο, τότε:

var c = b;
c === b; // true!
```

```
// Τέτοιο θέμα δεν υπάρχει
// με τα strings
var s = "test";
var t = "test";
s === t; // True
```

Επαναληπτικές Δομές Ελέγχου

- Όμοίως με γνωστές γλώσσες προγ/μού η Javascript προσφέρει:
    - `for` επαναλήψεις: για έναν γνωστό αριθμός φορών, π.χ. από μια αρχική ακέραια τιμή σε μια τελική.
    - `while`: "όσο ισχύει μια συνθήκη, επανάλαβε"
    - `do..while`: "επανάλαβε όσο ισχύει μια συνθήκη" (δηλ. έλεγχος στο τέλος) [αυτό σημαίνει ότι η επάναληψη θα γίνει τουλάχιστον 1 φορα!]
- Μεσα σε μια επανάληψη μια εντολή `break` προκαλεί τον (πρώιμο) τερματισμός των επαναλήψεων ενώ μια εντολή `continue` προκαλεί την παράλειιψη της τρέχουσας επανάληψης και την μεταπήδηση στην επόμενη.

Παράδειγμα: απλή επανάληψη στα στοιχεία ενός Array

```
for (let i=0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

Bug! Τι θα γίνει αν το array δεν έχει στοιχεία;;

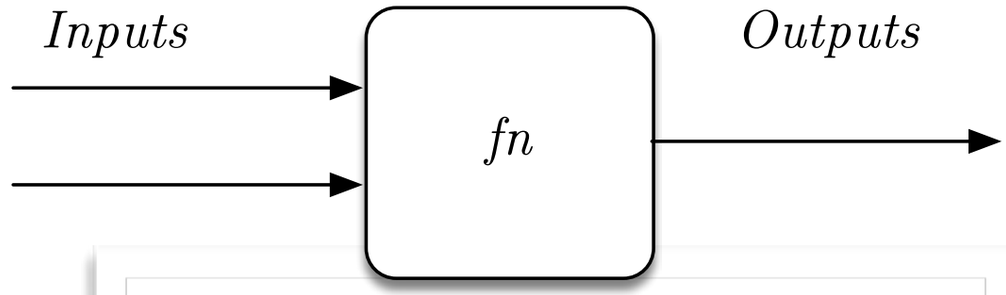
```
let i = 0;
while (i < arr.length) {
    console.log(arr[i]);
    i++;
}
```

```
let i = 0;
do {
console.log(arr[i]);
    i++;
} while (i < arr.length);
```

Συναρτήσεις

Δήλωση και κλήση συναρτήσεων: Γενικά

- Δήλωση της συνάρτησης με το keyword `function`
  - Παράμετροι μέσα σε παρενθέσεις
  - Εντολές συνάρτησεις μέσα σε αγκύλες
  - Το `return` προκαλεί την έξοδο από τη συνάρτηση και την επιστροφή της τιμής (αποτελέσματος) της συνάρτησης στο σημείο που κλήθηκε
- Κλήση της συνάρτησης με το όνομα της και τιμές εισόδου για τις παραμέτρους της

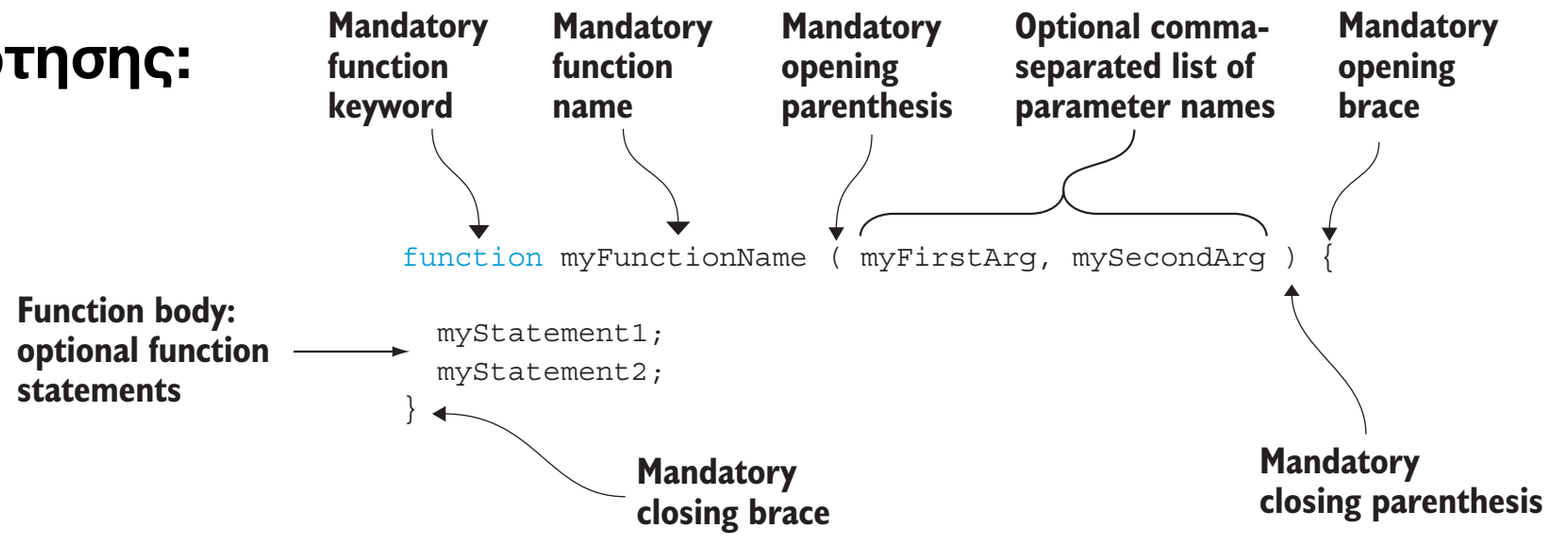



```
function is_even(n) {
    let rem = n % 2;
    return rem === 0;
}
function is_odd(n) {
    return !is_even(n);
}
```

```
var b = is_odd(933);
```

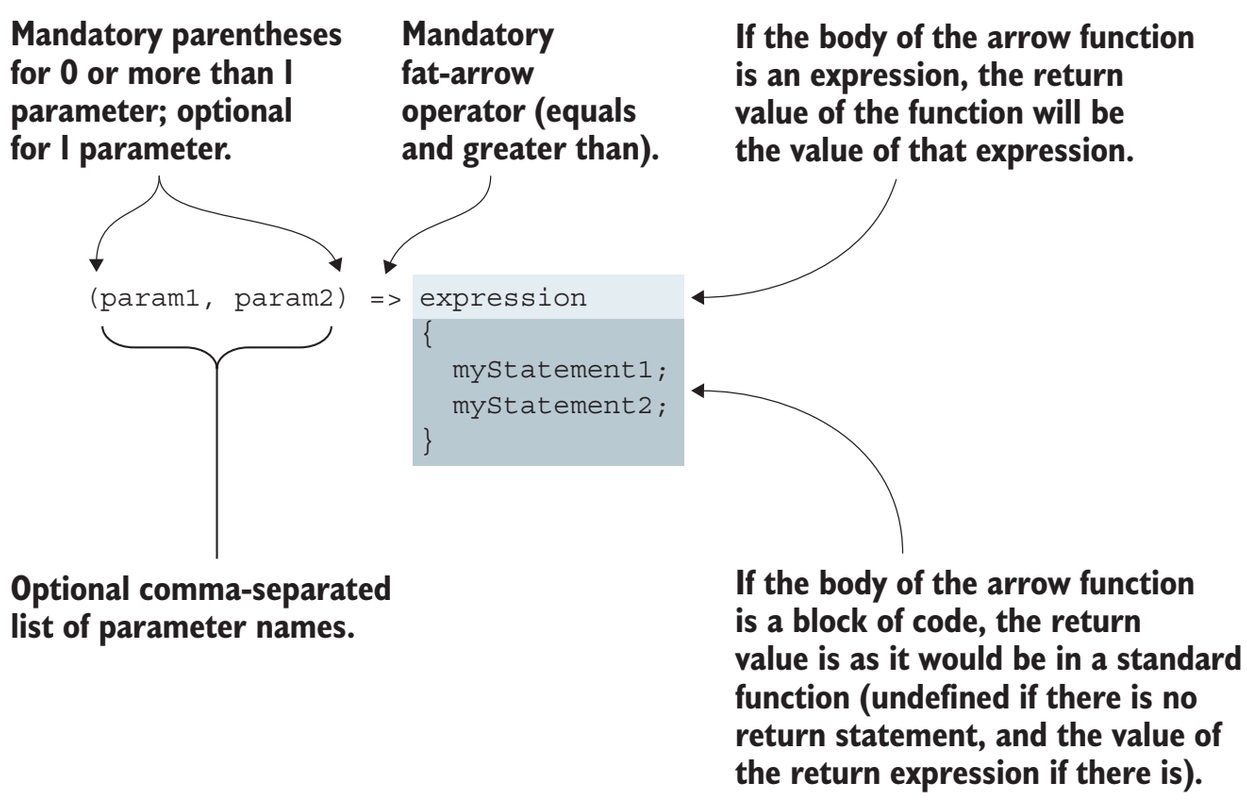
Συναρτήσεις (συν.)

Δήλωση συνάρτησης:



"Δήλωση" arrow function:

ES6



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Συναρτήσεις ως τιμές σε μεταβλητές

- Μπορούμε να δημιουργήσουμε "εκφράσεις συναρτήσεων" (*function expressions*) και να τις δώσουμε ως τιμές σε μεταβλητές

```
let is_even = function (n) {
    let rem = n % 2;
    return rem === 0;
}

let is_odd = function (n) {
    return !is_even(n);
}

let b = is_odd(933);
```

Συναρτήσεις

- Οι συναρτήσεις είναι "πολίτες 1ης κατηγορίας" στη Javascript: μπορούν να αποθηκευονται σε μεταβλητές, να επιστρέφονται από άλλες συναρτήσεις, και να δίνονται ως ορίσματα σε άλλες συναρτήσεις, κλπ

- Για παράδειγμα η μέθοδος `filter` των arrays παίρνει ώς παράμετρο μια άλλη συνάρτηση!

  - ..και επιστρέφει ένα νέο array που περιέχει τα στοιχεία του αρχικού για τα οποία η συνάρτηση-όρισμα επιστρέφει `true`

```
let words = ['spray', 'limit', 'elite', 'exuberant',
             'destruction', 'present'];

let filter_fn = function(word) {
    return word.length > 6;
}
let result = words.filter(filter_fn);

console.log(result); // ["exuberant", "destruction", "present"]

// Το ίδιο:
let result = words.filter(function(word) {
    return word.length > 6;
});
```

Συναρτήσεις

- Οι συναρτήσεις είναι "πολίτες 1ης κατηγορίας" στη Javascript: μπορούν να αποθηκευονται σε μεταβλητές, να επιστρέφονται από άλλες συναρτήσεις, και να δίνονται ως ορίσματα σε άλλες συναρτήσεις, κλπ

- "Arrow" functions (=>) για "μικρής κλίμακας" συναρτήσεις

ES6

```
let range = function(start, end) {
        let current = start;
        return function() {
            if (current < end)
                return current++;
            else
                return undefined;
        }
    }

let r = range(3, 5);

r(); // 3
r(); // 4
r(); // undefined
```

```
let r = [1,2,3,4,5,6,7,8,9];
r.filter(e => e % 2 == 1); // [ 1, 3, 5, 7, 9 ]
```

Άσκηση (Πίνακες, συναρτήσεις)

Να γραφεί μια συνάρτηση με όνομα `count_pairs` που να δέχεται τις εξής παραμέτρους:

- Ένα array με ακεραίους με όνομα παραμέτρου `arr`
- Έναν ακέραιο με όνομα παραμέτρου `n`

Η συνάρτηση θα υπολογίζει ζευγάρια στοιχείων του `arr` έχουν άθροισμα `n` και επιστρέφει το πλήθος τους.

Παράδειγμα χρήσης:

```
var arr = [1, 5, 7, -1];
var n = 6;

count_pairs(arr, n); // Επιστρέφει 2 διότι 1+5=6, και 7-1=6
```

Συναρτήσεις (συν.)

- Οι συναρτήσεις ως *first-class* objects μπορούν να είναι στοιχεία ενός πίνακα
- `Function.name`: το όνομα της συνάρτησης
- `Array.forEach`: εφαρμόζει μια συνάρτηση σε κάθε στοιχείο του πίνακα
- `Array.map`: εφαρμόζει μια συνάρτηση σε καθε στοιχείο του πίνακα **και** επιστρέφει ενα νέο πίνακα με τα αποτελέσματα
- `Array.filter:` εφαρμόζει μια συνάρτηση σε καθε στοιχείο του πίνακα και επιστρέφει ενα νέο πίνακα που περιέχει τα στοιχεία του αρχικού πίνακα για τα οποία η συνάρτηση επέστρεψε "αληθή" τιμή

```
function inc(a) {
    return a+1;
}
function double(a) {
    return 2*a;
}

let funs = [inc, double]
let arr = [1,2,3,4,5]

arr.map(function (i) {
    let d = {number: i};
    funs.forEach(f => d[f.name] = f(i));
    return d;
});
```

Callbacks

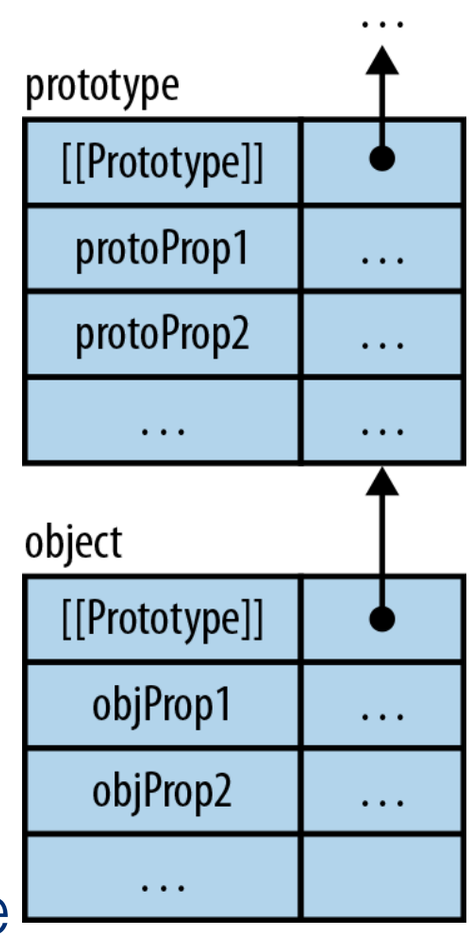
- Μια συνάρτηση που δίνεται ως όρισμα σε μια άλλη ονομάζεται **callback**
  - Είναι δηλαδή μια συνάρτηση που θα κληθεί αργότερα
- Η Javascript από κατασκευής υποστηρίζει "ασύγχρονο" προγραμματισμό με χρήση των callbacks
- Για παράδειγμα, η `setTimeout` έχει παράμετρο ενα callback function που θα κληθεί μετα από δοσμένο αριθμό milliseconds

```
setTimeout(() => {
    console.log("beep!");
}, 5000); // 5 seconds
```

Object-Oriented Javascript

Αντικείμενα

- Τα αντικείμενα είναι σχεδόν πανταχού παρόντα στην Javascript (ακόμα και οι συναρτήσεις ειναι αντικείμενα!)
- Μπορούν να δημιουργηθούν με την "κυριολεκτική" τους αναπαράσταση με αγκύλες (`{}`) ή με το `new Object` ή `Object.create`
- Κάθε αντικείμενο έχει ενα "prototype" ("πρωτότυπο") από το οποίο μπορεί να "κληρονομεί" δεδομένα και συναρτήσεις
- Επειδή όμως και το "πρωτότυπο" είναι αντικείμενο μπορεί να έχει ένα δικό του πρωτότυπο και ούτω καθεξής, δημιουργώντας έτσι μια αλυσίδα από πρωτότυπα ("prototype chain")
- Με την `Object.create(proto)` δημιουργούμε ενα αντικείμενο που έχει το δοσμένο `proto` ως prototype



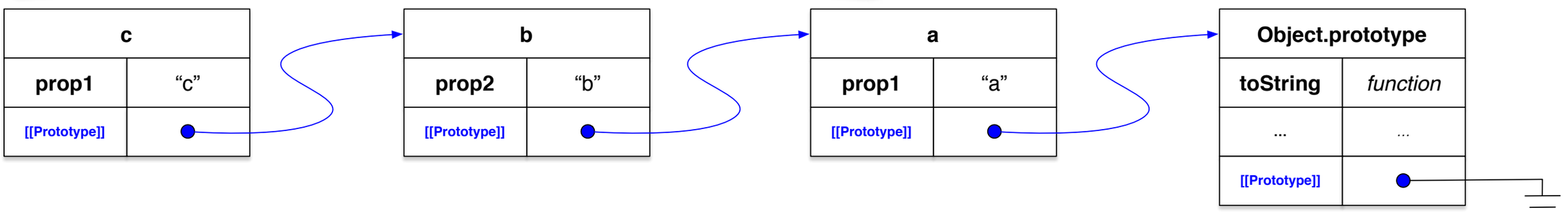
Παράδειγμα

- Όταν ζητάμε την τιμή ενός property σε ενα αντικείμενο, αυτή αναζητείται στην αλυσίδα με τα prototypes (ξεκινώντας από το ίδιο το αντικείμενο και πηγαίνοντας "προς τα πάνω") και η πρώτη που βρίσκεται αυτή επιστρέφεται

- Η αναζήτηση σταματάει στο prototype του Object που δεν έχει "παραπάνω" prototype

```
let a = {}; // Or: a = new Object();
a.prop1 = "a";

let b = Object.create(a);
b.prop2 = "b";

let c = Object.create(b);
c.prop1 = "c";
```

```
c.prop1 === "c"; // True
c.prop2 === "b"; // True
b.prop1 === "a"; // True
a.prop2 // undefined
```



Constructor Functions

- Οι constructors ειναι κανονικές συναρτήσεις που τις καλούμε με το `new` keyword.

- Κάθε αντικείμενο στην Javascript έχει μια ιδιότητα με το όνομα `constructor` που αναφέρει τη συνάρτηση (constructor function) που χρησιμοποιήθηκε για να δημιουργηθεί το αντικείμενο.

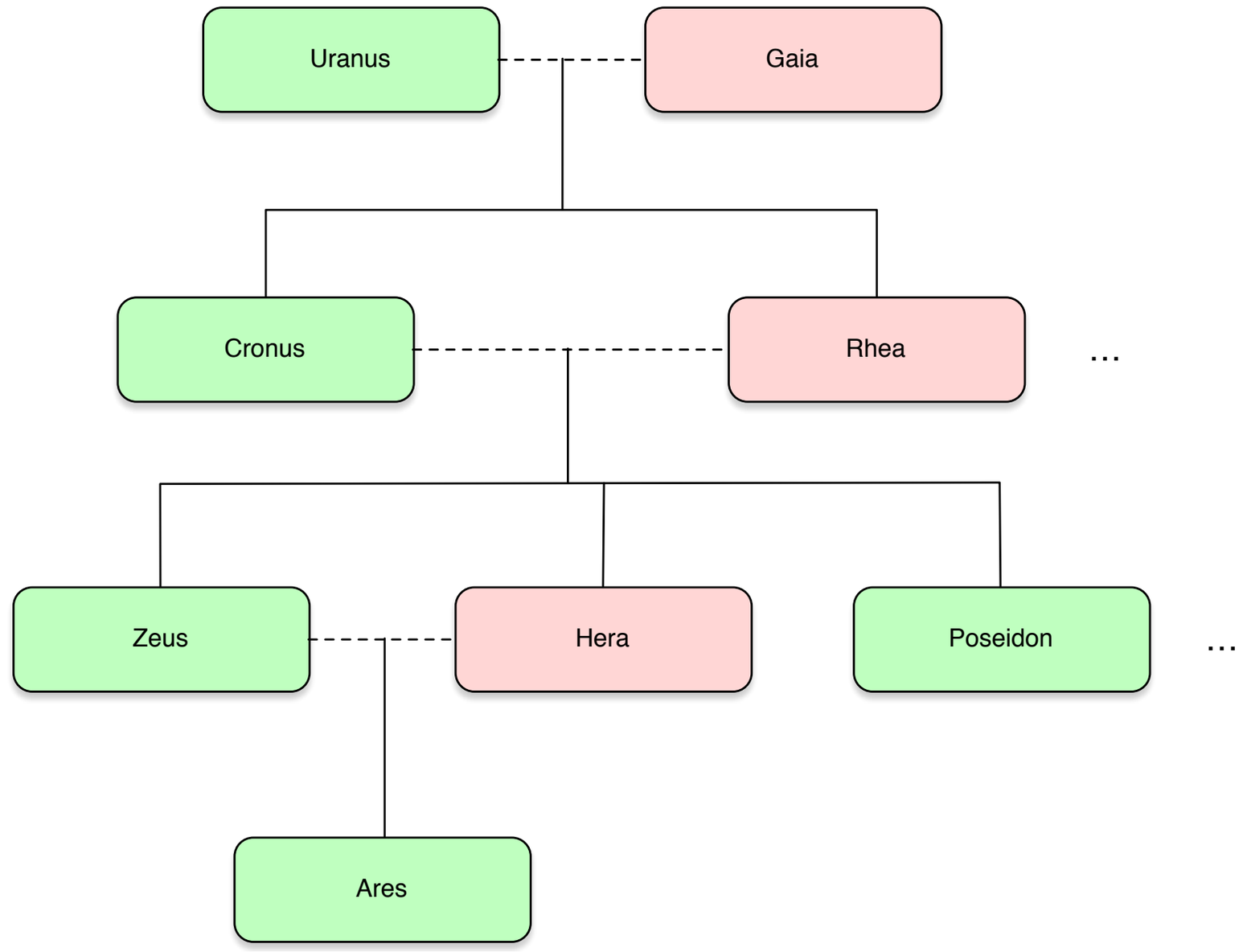
  - Ο τελεστής `instanceof` μπορεί να χρησιμοποιηθεί για να τσεκάρει αν ένα αντικειμενο δημιουργήθηκε από συγκεκριμένη constructor function

- Καθε συνάρτηση έχει μια ιδιότητα prototype που περιέχει τις κοινές ιδιότητες που έχουν τα αντικείμενα που δημιουργούνται με αυτή τη συνάρτηση όταν κληθεί με το `new` (δηλ. ως constructor function)

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

let p = new Person("Stelios")
p.constructor == Person; //  true
p instanceof Person; // true
```

Παράδειγμα: "Θεογονία"

- Αναπαράσταση (μέρους) του γενεαλογικού δέντρου των Ολύμπιων θεών
- Θα δούμε στη συνέχεια διάφορους τρόπους, αυξημένης πολυπλοκότητας αλλά και ευελιξίας



1. Απλές δομές - αντικείμενα

```
function makeDeity(name, father, mother)
{
    return {
        name : name,
        father : father,
        mother : mother,
        sayHi: function() {
            console.log(`Hi, I am ${name}`);
        }
    };
}

const uranus = makeDeity("Uranus", null, null);
const gaia = makeDeity("Gaia", null, null);
const cronus = makeDeity("Cronus", uranus, gaia);
const rhea = makeDeity("Rhea", uranus, gaia);
const zeus = makeDeity("Zeus", cronus, rhea);
const poseidon = makeDeity("Poseidon", cronus, rhea);
const hera = makeDeity("Hera", cronus, rhea);
const ares = makeDeity("Ares", zeus, hera);

ares instanceof makeDeity; // false
ares instanceof Object; // true
```

```
> zeus.sayHi()
Hi, I am Zeus
```

Το ίδιο με τη χρήση *constructor function*

- Με το constructor function `Deity` πάμε ενα βήμα πιο πέρα:

- Constructor ονομαζεται μια συνάρτηση που καλείται με το `new`

- Ενα constructor function θέτει properties στο `this` όπως φαινεται στο παράδειγμα δίπλα

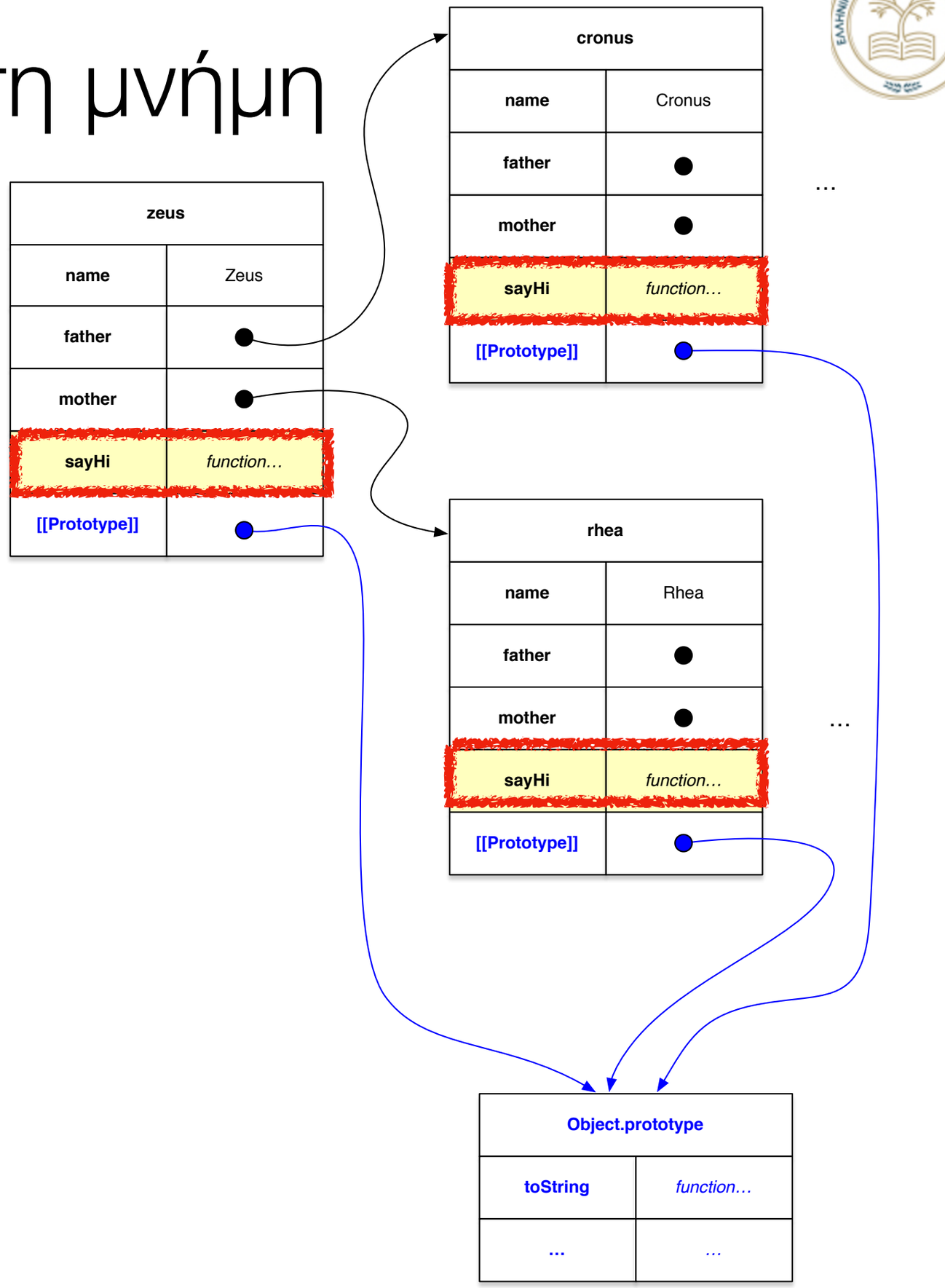
```
function Deity(name, father, mother)
{
    this.name = name;
    this.father = father;
    this.mother = mother;
    this.sayHi = function() {
        console.log(`Hi, I am ${this.name}`);
    }
}


const uranus = new Deity("Uranus", null, null);
const gaia = new Deity("Gaia", null, null);
const cronus = new Deity("Cronus", uranus, gaia);
const rhea = new Deity("Rhea", uranus, gaia);
const zeus = new Deity("Zeus", cronus, rhea);
const poseidon = new Deity("Poseidon", cronus, rhea);
const hera = new Deity("Hera", cronus, rhea);
const ares = new Deity("Ares", zeus, hera);

ares instanceof Deity; // true
ares instanceof Object; // true
ares.constructor === Deity; // true
```

Αναπαράσταση στη μνήμη

- Όλα τα αντικείμενα που φτιάχνονται με την `makeDeity` έχουν τα μονάδικα τους χαρακτηριστικά, δηλ δεν υπάρχει κάτι να τα ενώνει ώς ομάδα

  - *Εκτός*, από το ότι είναι αντικείμενα και έχουν ως κοινό το `Object.prototype` από το οποίο "κληρονομούν" π.χ. τη συνάρτηση `toString`

- Υπάρχει διπλός κώδικας, αφού το καθενα έχει τη δική του `sayHi` παρόλο που ο κώδικας αυτής της συνάρτησης είναι ακριβώς ο ίδιος σε όλα τα αντικείμενα



2. Χρήση του prototype

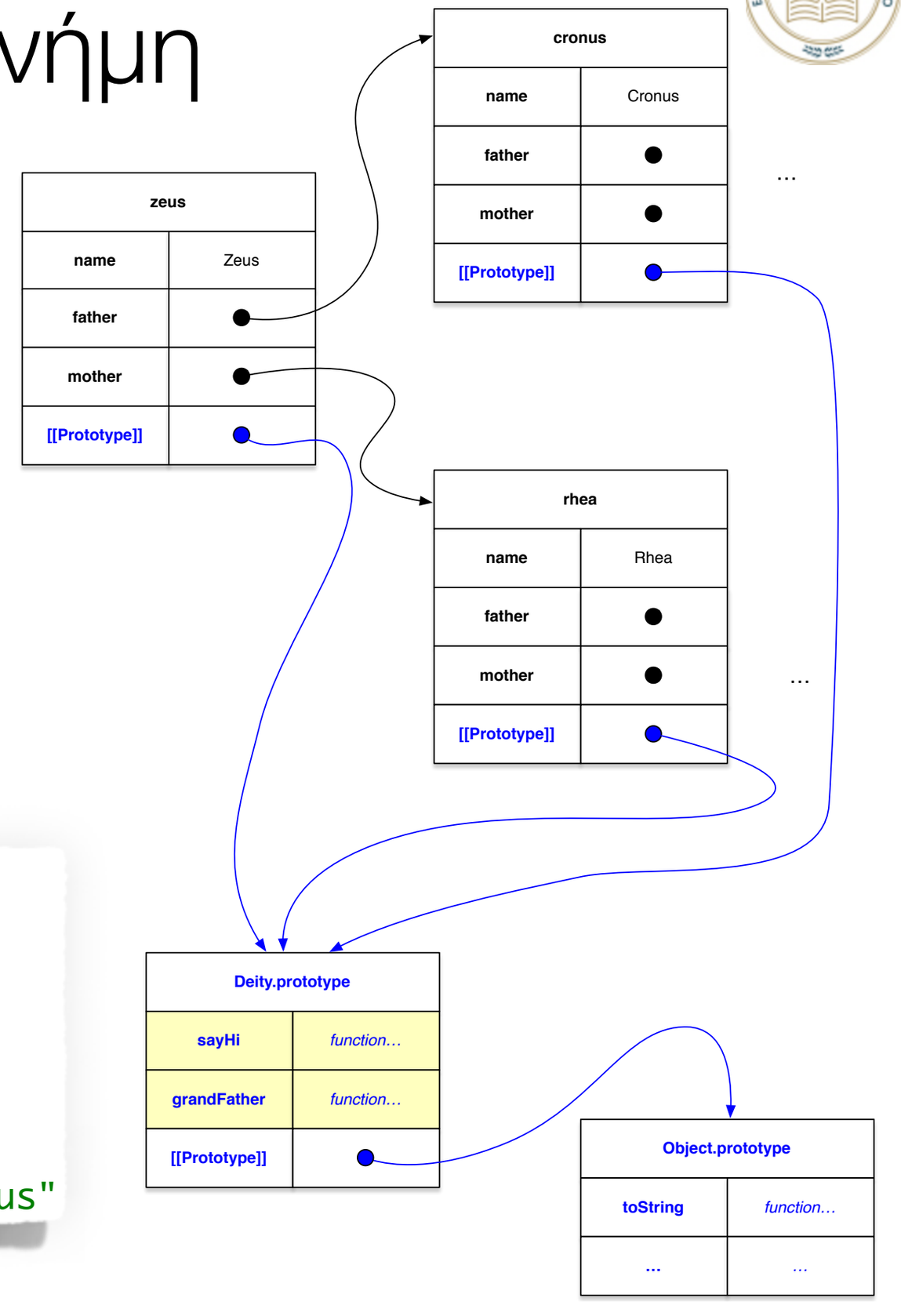
- `Deity.prototype` είναι το αντικείμενο που ορίζεται ως prototype σε καθε αντικείμενο που κατασκευάζεται όταν καλείται η συνάρτηση `Deity` ως constructor

- Όταν επομένως προσθέτουμε properties σε αυτό το prototype αντικείμενο, αυτές μοιράζονται ("κληρονομούνται") από όλα τα αντικείμενα που φτιάχνονται με τη συγκεκριμένη συνάρτηση

```
function Deity(name, father, mother)
{
    this.name = name;
    this.father = father;
    this.mother = mother;
}
Deity.prototype.sayHi = function() {
    console.log(`Hi, I am ${this.name}`);
}
const uranus = new Deity("Uranus", null, null);
const gaia = new Deity("Gaia", null, null);
const cronus = new Deity("Cronus", uranus, gaia);
const rhea = new Deity("Rhea", uranus, gaia);
const zeus = new Deity("Zeus", cronus, rhea);
const poseidon = new Deity("Poseidon", cronus, rhea);
const hera = new Deity("Hera", cronus, rhea);
const ares = new Deity("Ares", zeus, hera);
```

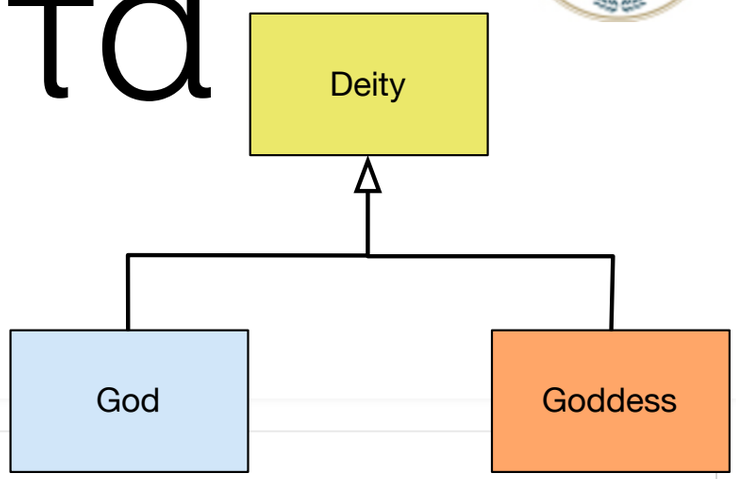
Αναπαράσταση στη μνήμη

- Το κοινό prototype μας επιτρέπει να προσθέσουμε κοινή λειτουργικότητα στα αντικείμενα, ακόμα και μετά τη δημιουργία τους




```
Deity.prototype.grandFather = function() {
    if (this.father != null)
        return this.father.father;
    else
        return null;
}
ares.grandFather().sayHi() // "Hi, I am Cronus"
```

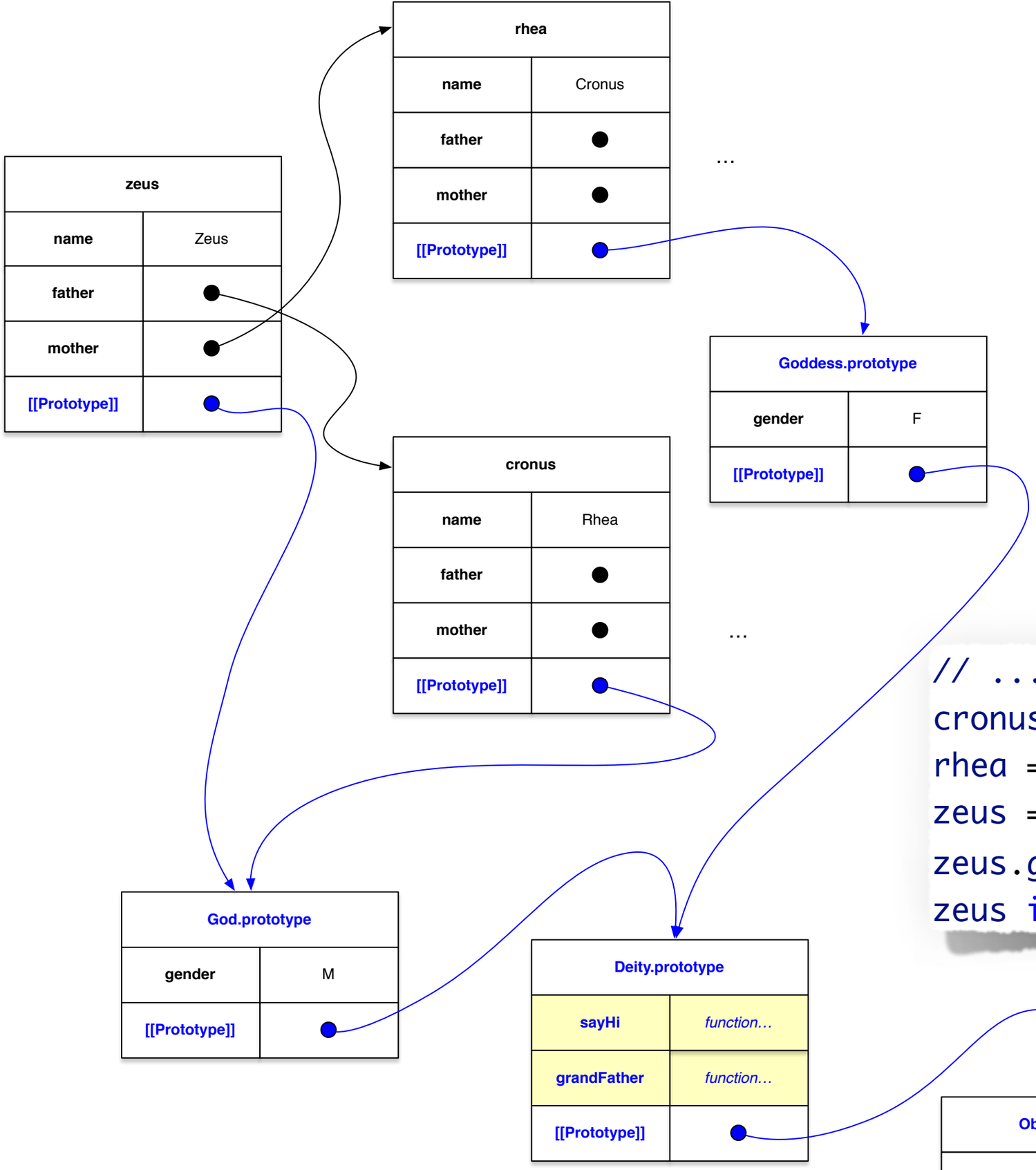
Κληρονομικότητα



- Ας προσπαθήσουμε να διαχωρήσουμε τις θεες (Goddess) από τους θεούς (God)

- Με έξυπνη χρήση των prototypes μπορούμε να το κάνουμε

```
function God(name, father, mother)
{
    Deity.call(this, name, father, mother);
}
God.prototype = Object.create(Deity.prototype);
God.prototype.gender = 'M';

function Goddess(name, father, mother)
{
    Deity.call(this, name, father, mother);
}
Goddess.prototype = Object.create(Deity.prototype);
Goddess.prototype.gender = 'F';

const uranus = new God("Uranus", null, null);
const gaia = new Goddess("Gaia", null, null);
// ...
```




```
// ...
cronus = new God("Cronus", uranus, gaia);
rhea = new Goddess("Rhea", uranus, gaia);
zeus = new God("Zeus", cronus, rhea);
zeus.gender === 'M'; // True
zeus instanceof God; // True
```

- Μεχρι εδώ έχουμε περιγραψει πώς δουλεύει η αντικειμενοστραφής Javascript σε χαμηλό επίπεδο
  - ..σε μια πολύ "*hackish*" προσέγγιση!

- Ευτυχώς η κατάσταση βελτιώθηκε με την έκδοση ECMAScript 2015 (ή ECMAScript 6)

3. Ορισμός κλάσεων στην ES6

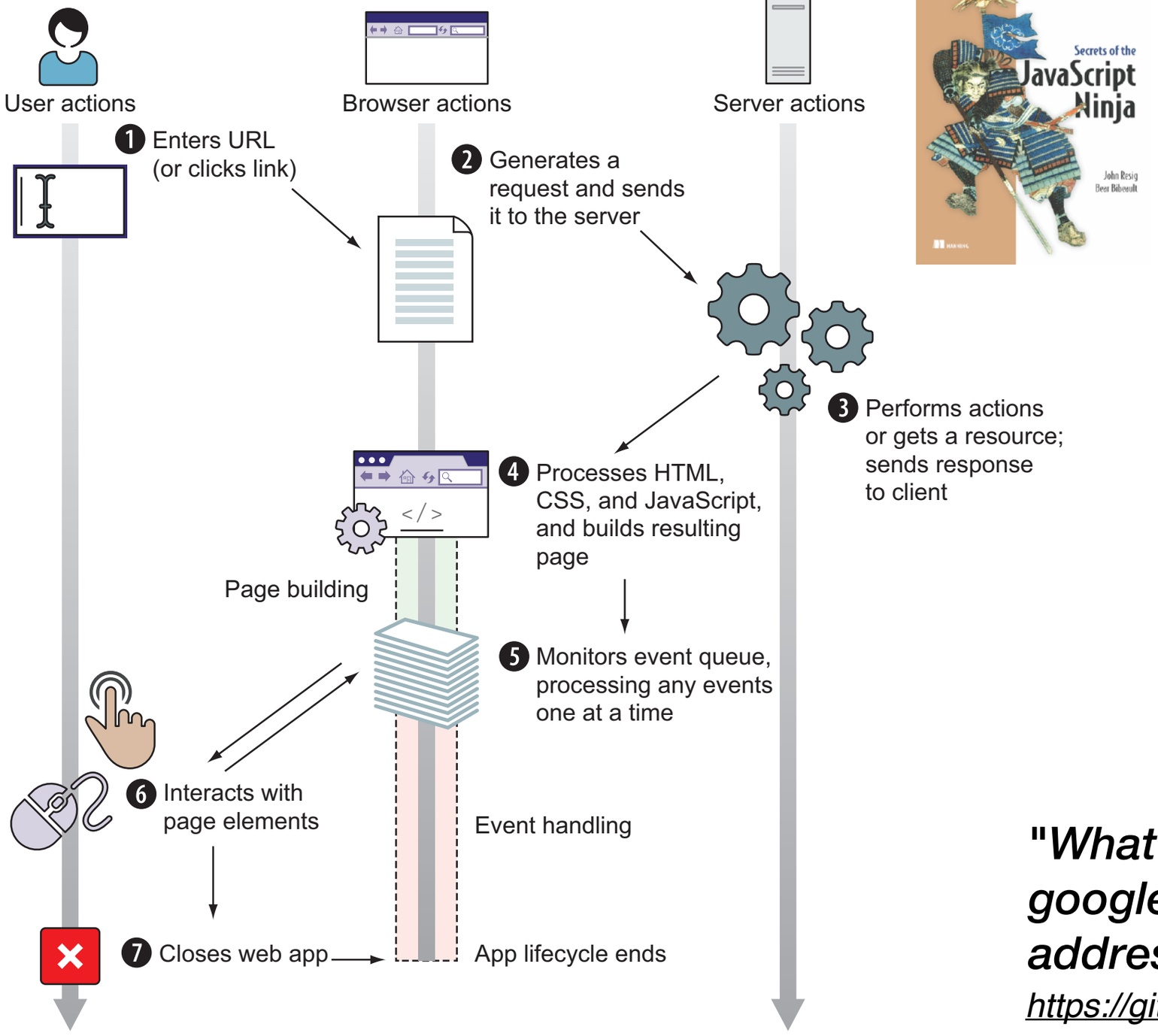
- Με παρόμοιο τρόπο όπως σε άλλες γλ. προγ/μού π.χ Java ή C++
- Είναι "*syntactic sugar*" δηλ. οι ορισμοί αυτοί μετατρέπονται σε κώδικα που ορίζει κατάλληλα τα prototypes..
  - Είναι απλώς μια πιο βολική σύνταξη

```
class Deity {
    constructor(name, father, mother) {
        this.name = name;
        this.father = father;
        this.mother = mother;
    }
    sayHi() {
        console.log(`Hi, I am ${this.name}`);
    }
}
class God extends Deity {
    constructor(name, father, mother) {
        super(name, father, mother);
        this.gender = 'M';
    }
}
class Goddess extends Deity {
    constructor(name, father, mother) {
        super(name, father, mother);
        this.gender = 'F';
    }
}
```

Κλήση του "πατρικού" constructor

Javascript και Web

Η γενική εικόνα



Σχετικό:

"What happens when you type google.com into your browser's address box and press enter?"

https://github.com/alex/what-happens-when

Javascript και HTML

- Ο κώδικας Javascript πρέπει να ειναι μέσα στο `<script>` tag

- Το `<script>` tag μπορεί να “αναφέρει” (κάνει reference με το `src` attribute) ένα ξεχωριστό αρχείο που περιέχει τον κώδικα:

```
<script src="../js/lib.js"></script>
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World in JavaScript</title>
</head>
<body>
    <script type="text/javascript">
        alert("Hello, world!");
    </script>
</body>
</html>
```

Θεση των scripts

- Τα Javascript scripts μπορούν να μπούν στο <head> ή/και στο <body>

  - Αν ειναι στο <head> τότε ο κώδικας (συνήθως/by default) “τρέχει” πριν να “φορτωθεί” η σελίδα

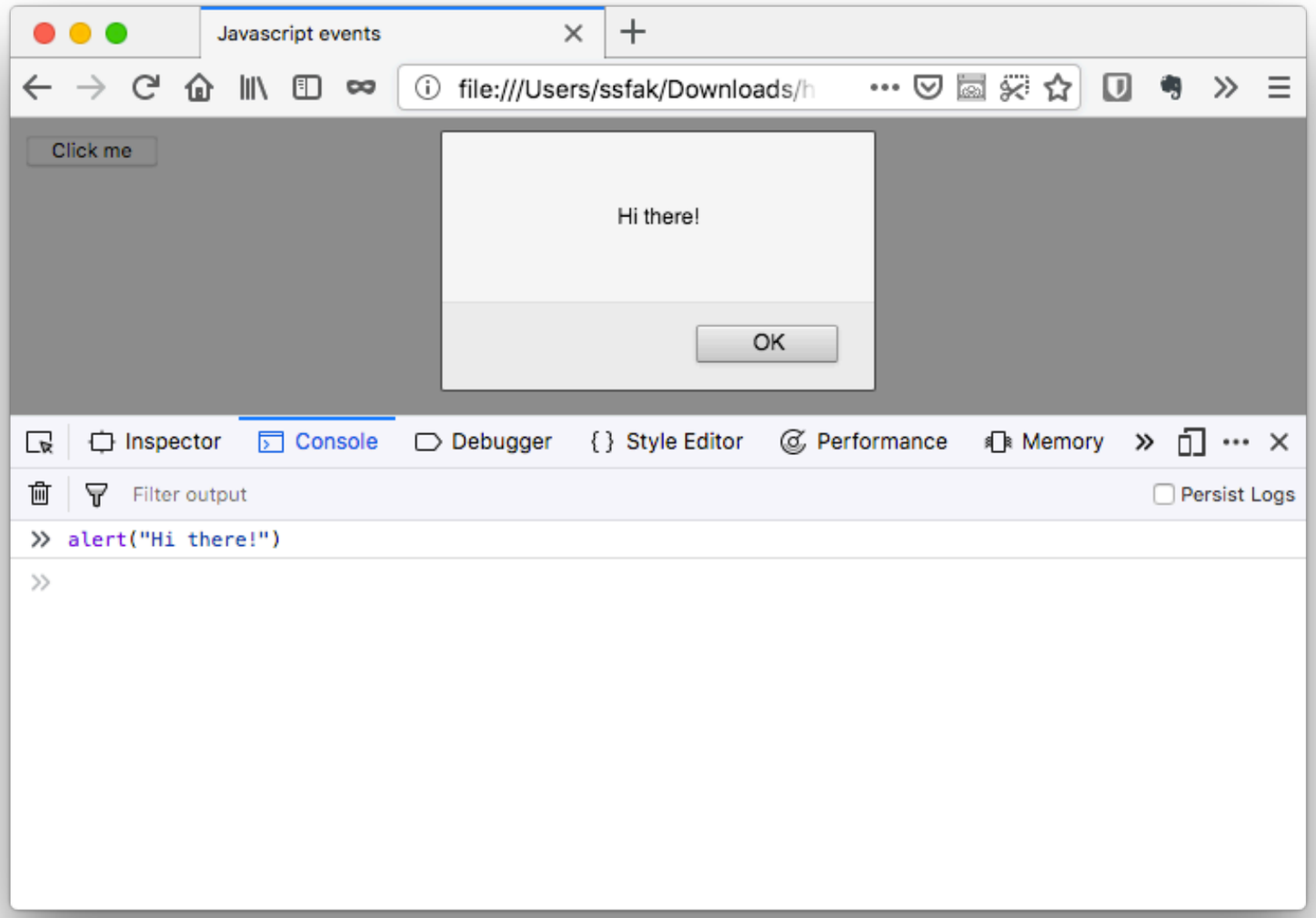
  - Αν ειναι στο <body> τότε ο κώδικας Javascript εκετελείται ενώ εμφανίζεται η σελίδα

- Συνηθίζεται σε <script> που ειναι στο <head> να βαζουμε ορισμούς συναρτήσεων και μεταβλητές που χρησιμοποιούνται στη συνέχεια από scripts που ειναι στο <body>

- Μπορούμε να έχουμε περισσότερα του ενός <script>

Browser developer tools

- Χρήσιμα για αποσφαλμάτωση:
  - `console.log` και `console.dir`
  - `alert` και `prompt`
- Εμφάνιση πληροφοριών για το DOM, stylesheets, επικοινωνία δικτύου, κλπ.
- Δυνατότητα τροποποίησης του περιεχομένου της σελίδας



Απλή επικοινωνία με τον χρήστη

- `alert(message)` : εμφανίζει το μήνυμα στο χρήστη

- `confirm(message)`: ζητά από το χρήστη να επιβεβαιώσει ή να ακυρώσει κάτι

- `prompt(msg, default)`: ζητά από το χρήστη να δώσει καποιο μήνυμα

Βασικά αντικείμενα με προγραμματιστική πρόσβαση

- `window` object: αναπαριστά την καρτέλα του πλοηγητή (browser tab) (π.χ. διαστάσεις)

- `navigator` object: η κατάσταση του browser (π.χ. προτιμώμενη γλώσσα χρήστη)

- `document` object: η σελίδα που βλεπουμε (HTML, CSS, ..) => **Document Object Model (DOM)**

Document Object Model

- Το DOM είναι μια ιεραρχική αναπαράσταση στη μνήμη του περιεχομένου μιας HTML (ή XML) σελίδας / εγγράφου

- Ορίζεται και μια "Προγραμματιστική Διεπαφή" (Programming Interface) για το χειρισμό των περιεχομένων

- Δηλ. με Javascript μπορούμε να χειριστούμε "δυναμικά" το περιεχόμενο, π.χ. να προσθέσουμε στοιχεία HTML, να αλλάξουμε τα περιεχόμενα τους κλπ

https://software.hixie.ch/utilities/js/live-dom-viewer/saved/6342

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Παράδειγμα σελίδας</title>
  </head>
  <body>
     <h1>Επικεφαλίδα</h1>
      <div>
        <img style="height:100px" src="https://media.giphy.com/...." alt="Bengal Tiger">
        <p>Μπορείτε να δείτε online το DOM στο <a href="https://...">Live DOM Viewer</a></p>
      </div>
  </
</ht
```

DOM Ιεραρχία

```
DOCTYPE: html
HTML
  HEAD
    #text:
    META charset="utf-8"
    #text:
    TITLE
      #text: Παράδειγμα σελίδας
    #text:
  #text:
  BODY
    #text:
    H1
      #text: Επικεφαλίδα
    #text:
    DIV
      #text:
      IMG style="height:100px" src="https://media.giphy.com/media/2Bkqvz4hcANy0/giphy-downsized.gif" alt="Bengal Tiger"
      #text:
      P
        #text: Μπορείτε να δείτε online το DOM στο
        A href="https://software.hixie.ch/utilities/js/live-dom-viewer/"
          #text: Live DOM Viewer
        #text:
      #text:
```

DOM Programming Interface

- Όλα τα HTML στοιχεία ορίζονται ως Javascript objects που έχουν ιδιότητες και μεθόδους
- Η ιεραρχία ξεκινάει από τη "ρίζα" που είναι το `window.document` object, και μετά ακολουθείται η δομή του HTML:
  - `window.document.head`
  - `window.document.body`

DOM object properties

- `nodeName` : το όνομα του στοιχείου (π.χ. "DIV") ή "#text" αν είναι κείμενο
- `textContent`: το περιεχόμενο κείμενο του στοιχείου (και όλων των στοιχείων που περιέχει)
- `innerHTML`: το HTML περιεχόμενο του στοιχείου
- Ιεραρχική δομή:
  - `parentNode` : γονικό στοιχείο,
  - `nextSibling`, `previousSibling` : επόμενος και προηγούμενος "αδελφός"
  - `firstChild`, `lastChild` : πρώτο και τελευταίο στοιχείο "παιδί"
  - `childNodes` : λίστα με παιδιά

Μεθοδοι "Εύρεσης" στοιχείων

Μέθοδος	Περιγραφή
`document.getElementById(id)`	Βρίσκει το στοιχείο με αυτό το id
`document.getElementsByTagName(name)`	Βρίσκει στοιχεία με αυτό το όνομα
`document.getElementsByClassName(name)`	Βρίσκει στοιχεία με αυτή το CSS class
`document.querySelector(selectors);`	Βρίσκει το πρώτο στοιχείο στο έγγραφο χρησιμοποιώντας τον CSS "επιλογέα" (selector) που δίνεται
`document.querySelectorAll(selectors);`	Βρίσκει μια λίστα με στοιχεία στο έγγραφο χρησιμοποιώντας τον CSS "επιλογέα" (selector) που δίνεται

```
<tbody>
    <tr class="row" id="first">
        <td>1.</td>
        <td>1st row</td>
    </tr>
    <tr class="row" id="second">
        <td>2.</td>
        <td>2nd row</td>
    </tr>
    <tr class="row" id="third">
        <td>3.</td>
        <td>3rd row</td>
    </tr>
</tbody>
```

```
document.getElementById("second");

document.getElementsByClassName("row");

document.querySelectorAll(".row > td:nth-child(2)")
```

Αλλάζοντας τα στοιχεία

Μέθοδος	Περιγραφή
`element.innerHTML = ...`	Αλλάζει το εσωτερικό HTML του στοιχείου
`element.setAttribute (attribute, value)`	Αλλάζει την τιμή ενός attribute του στοιχείου
`element.style.<property> = new style`	Αλλάζει το στυλ (style) ενός στοιχείου

- Π.χ. αν θέλουμε να "κρύψουμε" (να κάνουμε αόρατο) ένα element μπορούμε να γράψουμε:
  - `elem.style.display="none"`
  - `elem.style.display=""` για να ξαναγίνει "ορατό"

Προσθέτοντας και Αφαιρώντας Στοιχεία

Μέθοδος	Περιγραφή
`document.createElement(element)`	Δημιουργεί ένα HTML element
`document.removeChild(element)`	Αφαιρεί ("σβήνει") ένα HTML element
`document.appendChild(element)`	Προσθέτει ένα HTML element ως τελευταίο παιδί σε ένα άλλο
`document.replaceChild(newChild, oldChild)`	Αντικαθιστά ένα HTML element
`document.write(text)`	Γράφει το κείμενο στο HTML output stream

Javascript και "γεγονότα" (Events)

- Ένα Event είναι ένα σήμα ότι κάτι συνέβη. Όλα τα στοιχεία του DOM δημιουργούν τέτοια σήματα αλλά events δημιουργούνται και από αλλού (π.χ. timers)

- Μπορούμε να αντιδράσουμε στα events εγκαθιστώντας (κάνοντας register) έναν handler δηλ. μια Javascript συνάρτηση που τρέχει όταν συμβεί το event.

Παραδείγματα Events

- Πληκτρολόγιο:
  - `keydown`: ένα πλήκτρο πατήθηκε
  - `keyup`: ένα πλήκτρο ελευθερώθηκε
- Ποντίκι:
  - `click` / `dblclick`: το ποντίκι έκανε "κλικ" / "διπλό κλικ" σε ένα στοιχείο
  - `contextmenu`: "δεξί κλικ" σε ένα στοιχείο
  - `mousedown` / `mouseup`: όταν ένα κουμπί του ποντικιού πατήθηκε/ελευθερώθηκε
  - `mouseover` / `mouseout`: όταν το ποντίκι "μπαίνει"/"βγαίνει" στην περιοχή ενός στοιχείου
- Document events:
  - `DOMContentLoaded`: όταν η σελίδα HTML έχει φορτωθεί και το DOM έχει δημιουργηθεί

https://developer.mozilla.org/en-US/docs/Web/Events

Event handlers

- Υπάρχουν διάφοροι τρόποι να ορίσουμε ποιος θα είναι ο handler σε ένα event:

  - Σε ένα HTML attribute (`on..`), π.χ.
    ```
    <button onclick="alert('Hello')">
    ```

  - Σε Javascript θέτοντας την κατάλληλη attribute του στοιχείου
    ```
    let b = document.getElementById('btn');
    b.onclick = function() { alert ('Hello'); });
    ```

  - (**Προτιμότερο**) Σε κώδικα Javascript με το `addEventListener`

Σύνδεση HTML στοιχείου με event handler με το addEventListener

```
<body>
    <button id="btn">Click me</button>
</body>

<script type="text/javascript">
        var button = document.getElementById('btn');
        button.addEventListener('click',
                                function () {
                                    alert("You clicked me!");
                                });
</script>
```

- Το `addEventListener` έχει πρώτο όρισμα το event type (‘`click`’, ‘`load`’, … δηλ. χωρίς το `on`) και 2ο τη συνάρτηση (event handler/callback)

Παράδειγμα σύνδεσης μέσω attribute

```html
<!DOCTYPE html>
<html>

<head>
    <title>Javascript events</title>
</head>

<body>
    <h1>Welcome!</h1>
    <button onclick="message()">Press me!</button>
</body>

<script type="text/javascript">
    let count = 0;

    function message() {
        count++;
        alert(`You have pressed me ${count} times!`);
        document.querySelector("button").innerHTML =
            `Press me once more for ${count+1} presses`;
    }
</script>

</html>
```

Event Object

- Οι Event handlers έχουν πρώτο όρισμα (ανεξάρτητα αν το χρησιμοποιεί ο handler ή όχι) το `Event` object

- Το `Event` object περιέχει διάφορες γενικές πληροφορίες όπως:
  - `type`: π.χ. 'click'
  - `target`: το στοιχείο στο οποίο στάλθηκε το event (π.χ. Το στοιχείο στο οποίο εκανε click ο χρήστης)
  - `timeStamp`: πότε δημιουργήθηκε το event (σε milliseconds)
- ..και "ειδικές" πληροφορίες ανάλογα τον τύπο του

- Ποιό κουμπί στο ποντίκι πατήθηκε: `event.button` : 0 (αριστερό), 1 (μεσαίο), 2 (δεξί), ...

- Ποιό πλήκτρο πατήθηκε: `event.key`

- Τα keyboard και mouse events έχουν πληροφορία αν έχει πατηθεί και κάποιο modifier key (π.χ. το Shift):

  - `ctrlKey` , `shiftKey`, `altKey`, `metaKey`