Ανάπτυξη Κινητών Εφαρμογών

Εργαστήριο

Δημήτρης Ρίγγας

riggas@ionio.gr

Τμήμα Πληροφορικής

Ιόνιο Πανεπιστήμιο

Εργαστήριο 1 – Βασικές αρχές JavaScript

Στόχος: Να εξοικειωθείτε με τη σύγχρονη σύνταξη και λογική της JavaScript (ES6+).

JavaScript - προέλευση

Η `JavaScript` δημιουργήθηκε το 1995, από τη Netscape, ως ένας τρόπος να προστεθεί εκτελέσιμος κώδικας στις ιστοσελίδες και να τις κάνει *ζωντανές*.

Αρχικά ονομαζόταν LiveScript, αλλά λόγω της δημοφιλίας της Java την εποχή μετονομάστηκε σε JavaScript.
Δεν έχει όμως καμία άλλη σχέση με τη Java

Με τη διάδοσή της σε άλλους browsers (*και αφού αποτέλεσε σημαντικό πεδίο μάχης την περίοδο των browser wars*) προτυποποιήθηκε ως `ECMAScript`.

- ✓ Σήμερα οι όροι `JavaScript` και `ECMAScript` χρησιμοποιούνται συχνά ο ένας στη θέση του άλλου.

JavaScript - χαρακτηριστκά

Γλώσσα προγραμματισμού υψηλού επιπέδου

Διερμηνευόμενη (interpretted) γλώσσα

JavaScript δεν τρέχει μόνο μέσα στους browsers.

- ✓ εκτελείτε σε όποιο περιβάλλον περιλαμβάνει μια συμβατή `JavaScript engine`, πχ
  - ✓ V8 (Google), αξιοποιείται σε:
    - ✓ Chrome, Opera, Edge, κ.α.
    - ✓ Node.js, CouchDB, κ.α.
  - ✓ SpiderMonkey
    - ✓ FireFox
    - ✓ NEtScape (παλιότερα)
  - ✓ JavaScriptCore, Fourth Tier LLVM, Bare Bones Backend (B3)
    - ✓ Safari
    - ✓ WebKit browsers

- ✓ Βασίζεται σε **συμβάντα** και είναι **μονονηματικό**
- ✓ Το σύγχρονο JS χρησιμοποιεί **modules**, **arrow functions** και **promises**

.

(άνω τελεία)

τι θυμόμαστε από τις Τεχνολογίες Διαδικτύου;

(εδώ ας δούμε μερικά ενδιαφέροντα, μόνο)

Συναρτήσεις

- Μια συνάρτηση είναι ένα block κώδικα που μπορεί να κληθεί.
  - Δεν δηλώνει επιστρεφόμενο τύπο, ή αν επιστρέφει κάτι
  - Μπορεί να επιστρέφει κάτι, μπορεί και όχι
  - Μπορεί να λαμβάνει κάποιες παραμέτρους, προσδιορίζοντας μόνο τα ονόματά τους, όχι τύπους
  - Μπορεί να έχει όνομα, μπορεί όμως να είναι και ανώνυμη
  - Μπορεί να δωθεί ως παράμετρος σε άλλη μέθοδο

```
function hello(){   // Definition
    console.log("Hello");
}
hello();            // Call
```

```
function greetingStr(name){
    return "Hello "+name;
}
console.log( greetingStr("Eleni") );
```

```
let f = function(){
    console.log("This is anonymous");
}
f(); // prints: This is anonymous
let g = f;
g(); // prints: This is anonymous
```

```
let f = function(){
    console.log("This is anonymous");
}
(function(fun) {
    console.log("Calling another function");
    fun();
})(f);  // Defining and calling with a param
```

Συναρτήσεις & εμβέλεια

- Η χρήση `let` και `const` για την δήλωση μεταβλητών και σταθερών αυτές είναι local στο block που ορίζονται, αλλά προσοχή..
  - ό,τι ορίζεται σε ένα block είναι local εκεί

```
let x=10;
for(let i = 0; i < 10; i++){
    x += i;
}
console.log(x); // prints 55
typeof i;       // returns 'undefined'
```

  - μεταβλητές με το ίδιο όνομα σε εσωτερικό block κρύβουν άλλες σε πιο εξωτερικά

```
let x = 10;
function multiply(z){
    let x = 5;
    return x * z;
}
multiply(x); // prints 50, not 100
typeof z;    // returns 'undefined'
```

  - μια function αναζητά σε πιο εξωτερικά block

```
let x = 10;
function increase(){
    x +=1 ;
}
increase();
console.log(x); // prints 11
```
```js
```

  - Όλες οι συναρτήσεις θεωρείται ότι δηλώνονται στην κορυφή του αρχείου και το σημείο κλήσης τους δεν δημιουργεί πρόβλημα εμβέλειας

```
isInScope();
function isInScope(){
    console.log("This function is defined below
    its call??? :-/");
}
```


Συναρτήσεις arrow

- Οι συναρτήσεις μπορούν να οριστούν και με πιο σύντομη σύνταξη:

κλασσική σύνταξη:

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

multiline arrow function:

```
hello = () => {
    console.log("Hello");
}
```

single **statement** arrow function:

```
hello = () => console.log("Hello");
```

arrow function με παράμετρο:

```
hello = (name) => console.log("Hello "+name);
```

arrow function επιστρεφόμενη τιμή:

```
hello = () => "Hello World!";
```

ή, *για μονή παράμετρο*, ακόμη πιο απλά:

```
hello = name => console.log("Hello "+name);
```

Παράμετροι συναρτήσεων

- Αν κατά την κλήση μιας συνάρτησης δοθούν περισσότερες παράμετροι από αυτές που δηλώνει η συνάρτηση, αυτές αγνοούνται χωρίς μήνυμα λάθους.
- Μια συνάρτηση μπορεί να δηλώνει κάποιες παραμέτρους ως προαιρετικές, δίνοντας για καθεμία την τιμή που θα πρέπει να χρησιμοποιείται αν δεν παρέχεται κατά την κλήση
  - ή μπορεί εντός της συνάρτησης να υπάρχει έλεγχος για το εάν δόθηκε κάποια παράμετρο και αναλόγως να εκτελείται ο αντίστοιχος κώδικας

```
power = (num, exp=2) => num ** exp ;
power(3);     // returns 9
power(2, 10) // returns 1024
```

```
function minus(a, b) {
  if (b === undefined) return -a;
  return a - b;
}
minus(10);     // returns -10
minus(10, 5);  // returns  5
```

Πρότυπες συμβολοσειρές (Template Strings ή Literals)

Οριοθετούνται με χαρακτήρες backtick `` ` ``, επιτρέποντας συμβολοσειρές πολλαπλών γραμμών, παρεμβολή συμβολοσειρών με ενσωματωμένες εκφράσεις και ειδικές κατασκευές που ονομάζονται πρότυπα με ετικέτες.
Μερικά παραδείγματα:

```
`string text`

`string text line 1
 string text line 2`

`string text ${expression} string text`

tagFunction`string text ${expression} string text`
```

Πρότυπες συμβολοσειρές (Template Strings ή Literals)

Εκτύπωση συμβολοσειράς πολλαπλών γραμμών

```
console.log(`string text line 1
string text line 2`);
```

```
string text line 1
string text line 2
```

Ενσωμάτωση εκφράσεων (χωρίς concatenation)

```
const a = 5;
const b = 10;
console.log(`Fifteen is ${a + b} and not ${2 * a + b}.`);
```

```
Fifteen is 15 and not 20.
```

Πρότυπα με ετικέτες

```
const person = "Mike";
const age = 28;
function myTag(strings, personExp, ageExp) {
  const str0 = strings[0]; // "That "
  const str1 = strings[1]; // " is a "
  const str2 = strings[2]; // "."
  const ageStr = ageExp < 100 ? "youngster" : "centenarian";
  // We can even return a string built using a template literal
  return `${str0}${personExp}${str1}${ageStr}${str2}`;
}
const output = myTag`That ${person} is a ${age}.`;
console.log(output);
```

Το πρώτο όρισμα της συνάρτησης ετικέτας περιέχει μια σειρά από τιμές συμβολοσειρών.
Τα υπόλοιπα ορίσματα σχετίζονται με τις εκφράσεις/ παραμέτρους.

```
That Mike is a youngster.
```

Αποδόμηση

Η σύνταξη αποδόμησης καθιστά δυνατή την αποσυσκευασία τιμών από πίνακες ή ιδιοτήτων από αντικείμενα σε ξεχωριστές μεταβλητές.
Μπορεί να χρησιμοποιηθεί σε θέσεις που λαμβάνουν δεδομένα (όπως η αριστερή πλευρά μιας ανάθεσης ή οπουδήποτε δημιουργούνται νέες συνδέσεις αναγνωριστικών).

```
const user = { id: 7, username: "mark" };
const { id, username } = user;
console.log(`${username}'s ID is ${id}`)
```

```
mark's ID is 7
```

Ασύγχρονο JavaScript

Το JavaScript χρησιμοποιεί ένα **event loop**.
Ασύγχρονες λειτουργίες είναι οι **callbacks**, **promises** και **async/await**.

Το JavaScript εκτελείται σε ένα μόνο νήμα (single thread), που σημαίνει ότι μπορεί να κάνει μόνο ένα πράγμα κάθε φορά. Ωστόσο, καταφέρνει να χειρίζεται πολλαπλές εργασίες ταυτόχρονα, χάρη στον βρόχο συμβάντων.

- Στοίβα κλήσεων
  Παρακολουθεί ποια συνάρτηση εκτελείται εκείνη τη στιγμή.
- Web API / Εργασίες παρασκηνίου
  Συναρτήσεις όπως setTimeout, fetch ή event listeners μεταβιβάζονται στο Node.js API προς εκτέλεση και η επόμενη εργασία ξεκινά.
- Ουρά επιστροφής κλήσης (Callback Queue)
  Μόλις ολοκληρωθούν αυτές οι εργασίες παρασκηνίου, οι επιστροφές κλήσης τους τοποθετούνται σε μια ουρά.
- Βρόχος συμβάντων
  Ο βρόχος συμβάντων ελέγχει συνεχώς *Είναι κενή η στοίβα κλήσεων; Εάν ναι, παίρνει την πρώτη επιστροφή κλήσης από την ουρά και την ωθεί στη στοίβα για να εκτελεστεί.*

Ασύγχρονο JavaScript - callbacks

```
console.log("Start");

setTimeout(() => {
  console.log("Timeout");
}, 0);

console.log("End");
```

```
Start
End
Timeout
```

Ακόμα κι αν το setTimeout έχει οριστεί σε 0, εξακολουθεί να περνάει από τον βρόχο συμβάντων και εκτελείται μετά τον σύγχρονο κώδικα.

Ασύγχρονο JavaScript - promises

```
fetch("https://jsonplaceholder.typicode.com/users")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

```
[
 {
   id: 1,
   name: 'Leanne Graham',
   username: 'Bret',
   email: 'Sincere@april.biz',
   address: {
     street: 'Kulas Light',
     suite: 'Apt. 556',
     city: 'Gwenborough',
     zipcode: '92998-3874',
     geo: [Object]
   },
 }
 ...
]
```

- ✓ Το fetch() επιστρέφει μια υπόσχεση (Promise) που επιλύεται σε ένα αντικείμενο απόκρισης (Response) μόλις ολοκληρωθεί το αίτημα.
- ✓ Το .then() χειρίζεται το αντικείμενο Response που επιλύθηκε από την κλήση fetch().
- ✓ Το res.json() είναι μια μέθοδος που διαβάζει το ρεύμα απόκρισης και το αναλύει ως JSON. Επιστρέφει επίσης ένα Promise, το οποίο επιλύεται στα πραγματικά δεδομένα (σε αυτήν την περίπτωση, μια συστοιχία αντικειμένων χρήστη).
- ✓ Λαμβάνει τα αναλυμένα δεδομένα JSON και το console.log(data) εκτυπώνει τα δεδομένα στην κονσόλα.

Ασύγχρονο JavaScript - async/await

Τα async/await είναι συντακτικά στοιχεία που χρησιμοποιούνται όπου υπάρχουν Promises και κάνουν τον ασύγχρονο κώδικα πιο εύκολο να διαβαστεί και να γραφεί, σαν να ήταν σύγχρονος κώδικας.
Ίδια λειτουργία με πριν:

```
async function getUsers() {
  try {
    const res = await fetch("https://jsonplaceholder.typicode.com/users");
    const users = await res.json();
    console.log(users);
  } catch (err) {
    console.error(err);
  }
}
getUsers();
```

πιο κατανοητό συντακτικό

Χρήση modules

- Τα modules σας επιτρέπουν να χωρίσουμε τον κώδικα σε επαναχρησιμοποιήσιμα αρχεία χρησιμοποιώντας export και import.
- Τα modules βοηθούν στην οργάνωση και την ενθυλάκωση της λογικής — επιτρέποντας καθαρότερα, συντηρήσιμα και δοκιμασμένα τμήματα κώδικα.

math.js

```
export const add = (a,b) => a+b;
```

main.js

```
import { add } from "./math.js";
console.log(add(2,3));
```

Εκτέλεση με Node.js:

```
node --input-type=module main.js
```

Εγκατάσταση & προετοιμασία

1. Εγκαταστήστε το **Node.js** (v18 +) → https://nodejs.org/en/download
2. Επαληθεύστε την εγκατάσταση

```
node -v
npm -v
```

3. Δημιουργήστε ένα φάκελο εργασίας

```
mkdir lab1 && cd lab1
code .
```

💡 Άσκηση 1 – Μεταβλητές και τύποι

Γράψτε ένα σενάριο που:

1. Δηλώνει 3 μεταβλητές (`name` , `age` , `student`)
2. Τις εκτυπώνει χρησιμοποιώντας πρότυπες συμβολοσειρές
3. Χρησιμοποιεί `typeof` για να δείξει κάθε τύπο

💡 Άσκηση 2 – Πίνακες & Βρόχοι

Δημιουργήστε έναν πίνακα με τα 5 αγαπημένα σας τραγούδια.
Χρησιμοποιήστε:

- έναν βρόχο **for…of** για να εκτυπώσετε το καθένα
- `.map()` για να δημιουργήσετε έναν πίνακα με τίτλους σε κεφαλαία JavaScript Reference: .map()

💡 Άσκηση 3 – Συναρτήσεις

Γράψτε μια συνάρτηση `grade(score)` που:

- επιστρέφει `«Pass»` αν το σκορ είναι ≥ 50
- επιστρέφει `«Fail»` σε κάθε άλλη περίπτωση
  Δοκιμάστε την με διάφορες τιμές.

💡 Άσκηση 4 – Αντικείμενα & Αποδόμηση

Ορίστε ένα αντικείμενο `book` με `title`, `author`, `year`.
Χρησιμοποιήστε αποδόμηση για να εκτυπώσετε:

“ “🟦 *title* από *author* (*year*)” ”

💡 Άσκηση 5 – Async Fetch

Γράψτε μια συνάρτηση που ανακτά χρήστες από `https://jsonplaceholder.typicode.com/users` και εκτυπώνει μόνο τα ονόματά τους.

🚀 Mini-project: Μορφοποίηση δεδομένων JSON

Εργασία:
Δημιουργήστε ένα σενάριο `formatUsers.js` που:

1. Ανακτά χρήστες από το `https://jsonplaceholder.typicode.com/users`
2. Εξάγει το όνομα, το email και την πόλη
3. Τα εκτυπώνει όμορφα μορφοποιημένα:

```
👤 Leanne Graham
📧 Sincere@april.biz
🏙️  Gwenborough
-----------------
```

Πηγές

- Marijn Haverbeke, Eloquent JavaScript, 4rd edition (2024), CC BY-NC
  https://eloquentjavascript.net
- CSS basics από Mozilla MDN
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
- JavaScript.info
  https://javascript.info
- JSFiddle test bed
  https://jsfiddle.net