Τεχνολογίες Διαδικτύου

JavaScript

Δημήτρης Ρίγγας & Ελένη Χριστοπούλου

riggas@ionio.gr | hristope@ionio.gr

Τμήμα Πληροφορικής

Ιόνιο Πανεπιστήμιο

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.

Η JavaScript δεν τρέχει μόνο μέσα στους browsers!

- ✓ Οι MongoDB and CouchDB τη χρησιμοποιούν ως scripting και query language
- ✓ Το Node.js παρέχει τη δυνατότητα να προγραμματίσει κανείς σε desktop ή server περιβάλλον με τη γλώσσα JavaScript

JavaScript - τι μπορεί να κάνει εντός του browser

μπορεί να

- προσθέσει & τροποποιήσει υπάρχον περιεχόμενο και στυλ
- ανταποκριθεί σε συμβάντα (πχ mouse clicks, pointer movements, key presses)
- πραγατοποιήσει αιτήματα και να μεταφέρει περιεχόμενο
- θέτει και διαβάζει cookies, κάνει ερωτήματα στο χρήστη, εμφανίζει μηνύματα
- αποθηκεύει δεδομένα στον browser (local storage)

δεν μπορεί να

- έχει πρόσβαση στο file system ή σε υπηρεσίες του λειτουργικού συστήματος
  - μπορεί να παρέχεται με ρητή εξουσιοδότηση από το χρήστη
- να έχει πρόσβαση σε άλλα tabs ή windows του browser
- να πραγματοποιεί HTTP κλήσεις σε άλλα domains (εκτός αυτού της σελίδας που είναι ήδη φορτωμένη στο browser)
  - μπορεί να καρακαμφθεί με κατάλληλα CORS headers

JavaScript - άλλες γλώσσες βασισμένες στη JavaScript

CoffeeScript

- γλώσσα προγραμματισμού η οποία μεταγλωττίζεται σε JavaScript
- συντακτικό εμπνευσμένο από Ruby, Python
- υποστηριζόμενη στη Ruby on Rails

TypeScript

- γλώσσα προγραμματισμού βασισμένη στη JavaScript
- strongly-typed, σε αντίθεση με την JavaScript
- υπερσύνολο της JavaScript

Flow

- γλώσσα προγραμματισμού βασισμένη στη JavaScript
- strongly-typed, σε αντίθεση με την JavaScript
- *Facebook*

Dart

- γλώσσα προγραμματισμού η οποία εκτελείται σε περιβάλλοντα εκτός browser, όπως mobile apps
- μεταγλωττίζεται σε JavaScript
- *Google*

JavaScript - *(τελικά,)* τι είναι;

Γλώσσα προγραμματισμού υψηλού επιπέδου

Διερμηνευόμενη (interpretted) γλώσσα

- 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

Τύποι δεδομένων

1. String
   ```
   let msg = 'Hello JS';
   ```
2. Number, τόσο για int όσο και float
   ```
   let num = 5;
   let num = 4.5;
   ```
3. Bigint, για αριθμούς εκτός του εύρους $\pm(2^{53}-1)$
   ```
   let x =
   BigInt("12345678901234567890123456789O");
   ```
4. Undefined
   ```
   let val;
   alert(val); // shows "undefined"
   ```
5. Boolean
   ```
   let checked = false;
   ```
6. Null, δεν αποτελεί "reference to a non-existing object" ή "null pointer". όπως σε άλλες γλώσσες, είναι απλά μια ειδική τιμή που αναπαριστά το "nothing", "empty" ή "value unknown".
7. Object, συλλογές ιδιοτήτων & τιμέων τους
   ```
   let person = {firstName:"Eleni",
   lastName:"Christopoulou"}
   ```
8. Symbol, δημιουργεί μοναδικά αναγνωριστικά αντικειμένων

Η JavaScript είναι `dynamically typed`, υπάρχουν τύποι δεδομένων, αλλά οι μεταβλητές δεν ορίζονται σε συγκεκριμένο τύπο και κατά την εκτέλεση του κώδικα μπορούν να μεταβαίνουν σε άλλος τύπους.
Παράδειγμα (αποδεκτό):

```
let n = 123;
n = 12.345;
n = "hello";
```

Μεταβλητές & σταθερές

1. Χρήση λέξης κλειδί `let`
   εισαγωγή το 2015
   `let newVar = 10;`
   `newVar = 2* newVar;`
2. Χρήση λέξης κλειδί `const` => σταθερά
   εισαγωγή το 2015
   `const newConst = 100;`
   `newVar += newConst;`
   `newConst = 0;` **Uncaught TypeError: Assignment to constant variable.**
3. Χρήση λέξης κλειδί `var`
   παλιότερος τρόπος δήλωσης, ισχύει από το 1995 **δεν συστήνεται**
   `var myVar = 5;` δήλωση μεταβλητής
   `myVar = 7;` τροποποίηση τιμής μεταβλητής
4. Καμία λέξη κλειδί :-)
   `sum = myVar + 2;` δημιουργία μεταβλητής και απόδοση τιμής

- Ονοματοδοσία μεταβλητών:
  - μπορεί να περιλαμβλανει unicode γράμματα (αν και δε συνίστανται οι non-latin χαρακτήρες), αριθμητικά ψηφία ή τα σύμβολα `$` και `_`, αλλά το πρώτο γράμμα δεν μπορεί να είναι αριθμητικό ψηφίο
  - είναι `case sensitive`
  - δεν επιτρέπονται οι δεσμευμένες λέξεις της γλώσσας
- Δοκιμάστε τους πιο πάνω κώδικες στη `Developer Tools` > `JavaScript console` ενός browser.
  - δείτε τις τιμές μεταβλητών εκτυπώνοντάς τες με `console.log(<variable name>)`

Δομή προγράμματος

- Δηλώσεις (statements), μπορεί να περιλαμβάνουν:
  - τιμές, τελεστές, εκφράσεις (expressions), λέξεις κλειδιά, σχόλια
- Semicolons `;`
  - τερματίζουν τις δηλώσεις
    `console.log('Hello');`
  - αν υπάρχουν, πολλές δηλώσεις μπορεί να υπάρχουν σε μια γραμμή

```
console.log('Hello');
console.log('World');
```

  - αν παραλείπονται... η JavaScript Engine προσπαθεί (best effort) να ερμηνεύσει ορθά τον κώδικα

```
console.log('Hello' +
            'World');
```

- Σχόλια
  - μίας γραμμής `// σχόλιο μίας γραμμής`
  - πολλών γραμμών

```
/* σχόλιο
πολλών
γραμμών */
```

- Εμβέλεια
  - πριν το 2015
    - `global` μεταβλητές ή
    - `function scope` μεταβλητές
  - μετά το 2015 (ECMAScript 6)
    - `block scope` με χρήση `let` ή `const`

Τελεστές

1. Εκχώρησης
   - `=`
   - `+=` συντομογραφία, πχ `x+=y` αντί `x=x+y`
   - ανάλογα `-= *= /= %= **=`
     - ανάλογα και για logical, shift, bitwise
2. Αριθμητικοί
   - `+ - * /`
   - `**` ύψωση σε δύναμη
   - `%` υπόλοιπο διαίρεσης
   - `++ --` αύξησης, μείωσης, πχ `y--` αντί `y=y-1`
3. Συνένωσης
   - `x = 'hello' + 5 + 5` αποδίδει την τιμή `'hello55'` στη μεταβλητή
   - αλλά, προσοχή στη σειρά: το `x = 5 + 5 + 'hello'` αποδίδει `10hello`
4. Σύγκρισης
   - `> < >= <=`
   - `==` προσοχή:
     - `5 == 6-1` αποτιμάται `true`
     - `'5' == 6-1` αποτιμάται `true`
     - `'5' == '6-1'` αποτιμάται (βέβαια) `false`
   - `===` ίση τιμή και ίδιος τύπος:
     - `5 === 6-1` αποτιμάται `true`
     - `'5' === 6-1` αποτιμάται `false`
     - `'5' === '6-1'` αποτιμάται `false`
   - ανάλογα για `!=` και `!==`
   - `?` τριαδικός τελεστής
     - ```
       x = 5
       y = (x>5) ? 'large' : 'small'
       ```
       ```

Τελεστές (συν.)

5. Λογικοί
   - `&& || !` and, or not
6. Τύπου
   - `typeof` επιστρέφει τον τύπο ενός αντικειμένου
     `typeof 'Text'` επιστρέφει `string`
     `typeof 5` επιστρέφει `number`
     `typeof 5.2` επιστρέφει `number`
     `typeof Math.PI` επιστρέφει `number`
     `typeof Math` επιστρέφει `object`
   - `instanceof` επιστρέφει `true` / `false` εάν μια μεταβλητή ανήκει σε ένα τύπο
7. Bitwise τελεστές
   προσοχή, οι τελεσταίοι μετατρέπονται σε signed 32-bit αριθμούς
   - `&` bitwise AND
   - `|` bitwise OR
   - `~` bitwise NOT
   - `^` bitwise XOR
   - `<<` bitwise shift left
     πχ `5 << 2` αποτιμάται σε 20 *δεκαδικό, όχι δυαδικό*
   - `>>` bitwise shift right
     πχ `5 >> 2` αποτιμάται σε 10 *δεκαδικό, όχι δυαδικό*
   - `>>>` bitwise shift right (unsigned)

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
```

  - 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
```

Πίνακες (Arrays)

- Μία μεταβλητή που μπορεί να λάβει περισσότερες από μία τιμές
  - οποιουδήποτε βέβαια τύπου
- Δήλωση πίνακα:

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
let fruits = new Array("Banana", "Orange", "Apple",
"Mango");
```

- Πρόσβαση στοιχείου πίνακα:

```
console.log(fruits[0]);
```

- Τροποποίηση στοιχείου πίνακα:

```
fruits[0] = "Cherry";
```

- Πλήθος στοιχείων (length **property**):

```
console.log(fruits.length); //χωρίς παρενθέσεις
```

- Looping κλασσικό:

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

- Looping με `forEach`:

```
fruits.forEach( (f) => console.log(f) );
```

- Looping με `for .. of`:

```
for (f of fruits) console.log(f);
```

Πίνακες (Arrays) - μέθοδοι

- Προσθήκη στοιχείων, οπουδήποτε με χρηση index

```
fruits[6]="Lemon";
```

- **προσοχή**: δημιουργεί κενά στον πίνακα και δημιουργείται πρόβλημα στην πρόσβαση με κλασσικό loop (infinite loop)

- Προσθήκη στοιχείων
(append στο τέλος):

```
fruits.push("Lemon");
```

- Εξαγωγή πρώτου στοιχείου (το οποίο επιστρέφεται):

```
let f = fruits.shift();
```

- Εξαγωγή τελευταίου στοιχείου
(το οποίο επιστρέφεται):

```
let f = fruits.pop();
```

- Εισαγωγή πρώτου στοιχείου και μετακίνηση +1 index όλων των υπαρχόντων:

```
fruits.unshift("Watermelon");
```

Πίνακες (Arrays) - μέθοδοι

- Διαγραφή n-οστού στοιχείου (**προσοχή**: αφήνει κενά):

```
delete fruits[2];
```

- Διαγραφή n-οστού στοιχείου (χωρίς να αφήνει κενά):

```
fruits.splice(2, 1);
```

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

Αντικείμενα (Objects)

- Στην ουσία είναι συλλογές από ιδιότητες (και τις αντίστοιχες τιμές τους).

```
let person = {
    firstName:"Eleni",
    lastName:"Christopoulou",
    email:"hristope@ionio.gr"
};
```

- Οι τιμές των ιδιοτήτων μπορούν να αλλάζουν, αλλά και νέες ιδιότητες μπορούν να προστίθενται, ο οποίες μπορεί είναι primitive τιμές, πίνακες, αντικείμενα, ακόμη και συναρτήσεις:

```
person.fullname = function(){ return this.firstName + " " + this.lastName};
```

- Το `console.log(person.fullname());` πλέον εμφανίζει `Eleni Christopoulou`
- Εντός ενός αντικειμένου, το `this` αναφέρεται στο τρέχον αντικείμενο
- Η πρόσβαση στις τιμές ιδιοτήτων ενός αντικειμένου γίνεται είτε με χρήση του τελεστή `.` ή σαν το αντικείμενο να είναι ένα associative array:
  - Τόσο το `person.email`, όσο και το `person['email']` επιστρέφουν την τιμή `hristope@ionio.gr`

Αντικείμενα (Objects) - ιδιότητες

- Πρόσβαση στις ιδιότητες ενός αντικειμένου παρέχει η build-in συνάρτηση `Objects.keys()`:

```
console.log(Object.keys(person));
```

επιστρέφει ένα πίνακα με τα κλειδιά (ονόματα ιδιοτήτων) του αντικειμένου person:
`["firstName", "lastName", "email", "fullname"]`

- Απαρίθμηση τιμών ιδιοτήτων ενός αντικειμένου:

```
Object.keys(person).forEach(function (p){console.log(p+": "+person[p])});
```

εμφανίζει:

```
firstName: Eleni
lastName: Christopoulou
email: hristope@ionio.gr
fullname: function(){ return this.firstName + \&quot; \&quot; + this.lastName}
```

- Όμοιο αποτέλεσμα δίνει και η πρόσβαση με `for .. in`:

```
for (p in person) console.log(p+": "+person[p]);
```

Control

- Τα συνήθη `if`, `for`, `while`, ...
- και επιπλέον δομές, όπως `forEach`, `for .. of`, `for .. in` που είδαμε ήδη

“ *learn by doing* ”

μπορούμε να μάθουμε όλη τη JavaScript, *σήμερα*;

ας εστιάσουμε σε

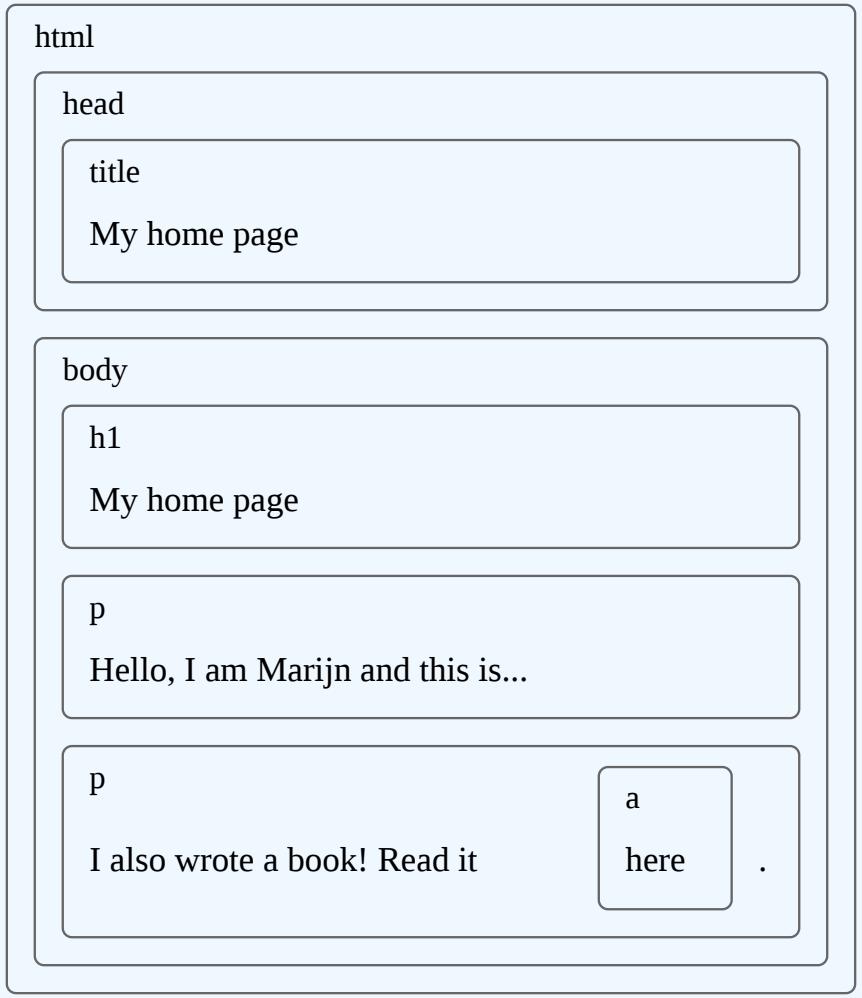
JavaScript & Browsers

Document Object Model

- Όταν φορτώνεται μια ιστοσελίδα σε ένα browser αυτός σχηματίζει τη δομή της ως ένα μοντέλο, το Document Object Model.
  Πχ για την ιστοσελίδα:

```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```

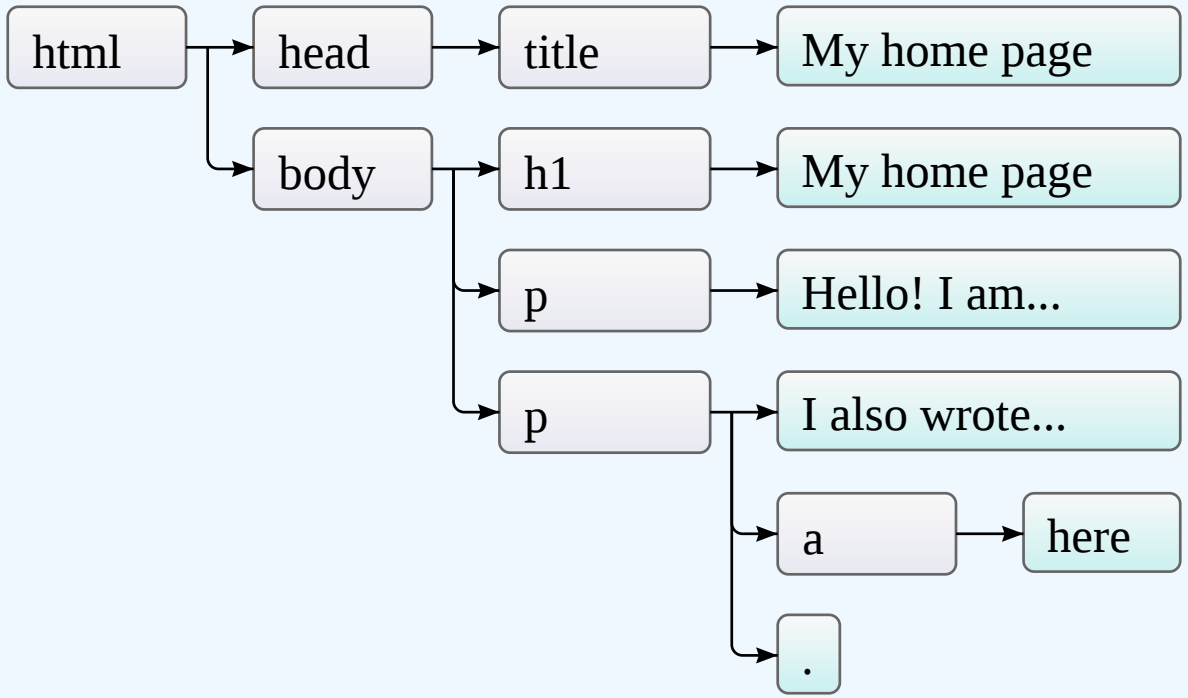
- δημιουργείται το DOM στα δεξιά. Κάθε κουτί αντιστοιχεί σε ένα αντικείμενο με το οποίο μπορούμε προγραμματιστικά να αλληλεπιδράσουμε ρωτώντας ποια άλλα αντικείμενα περιέχει, ποιο κείμενο περιέχει, ποιο το γονικό του αντικείμενο, κ.α.



https://eloquentjavascript.net/14_dom.html

Δενδροειδής δομή DOM

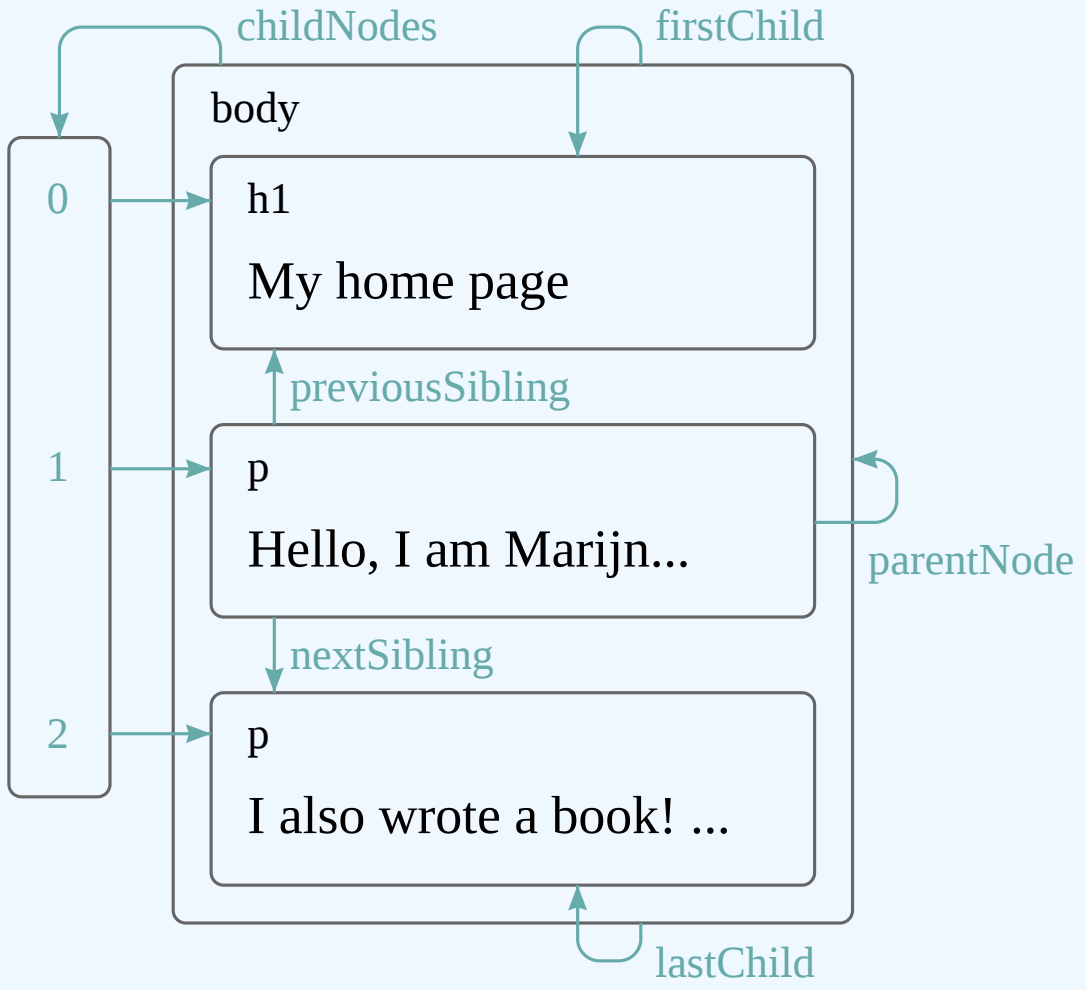
- Το DOM μπορεί να αναπαρασταθεί και ως ένα δέντρο
- Το διάγραμμα οπτικοποιεί τις σχέσεις γονέα-απογόνων, που εντάσονται τα στοιχεία απλού κειμένου, ποια στοιχεία έχουν κοινό γονέα, κ.α.



https://eloquentjavascript.net/14_dom.html

Συνδέσεις μεταξύ κόμβων στο DOM

- Οι κόμβοι που δημιουργούνται στο DOM περιλαμβάνουν πολλούς *δείκτες*, τους οποίους μπορούμε να προσπελάσουμε μέσω της JavaScript όπως θα δούμε
- Για κάθε κόμβο μπορούμε να δούμε (μεταξύ άλλων):
  - τους άμεσους απογόνους τους
    Η `childNodes` είναι μια **ιδιότητα** κάθε κόμβου στο DOM που επιστρέφει μια read-only `NodeList`, δλδ μια λίστα DOM κόμβων, με τους κόμβους απογόνους του τρέχοντος στοιχείου
  - κόμβους πριν ή μετά το τρέχοντα
    Η `previousSibling` και η `nextSibling` είναι **ιδιότητες** του τρόχοντος κόμβου που επιστρέφουν τον αμέσως επόμενο ή προηγούμενο DOM κόμβο στο ίδιο επίπεδο του DOM γράφου
  - κ.ο.κ.



https://eloquentjavascript.net/14_dom.html

Πλήρης λίστα ιδιοτήτων και μεθόδων:

https://www.w3schools.com/jsref/dom_obj_all.asp

Αναζήτηση κόμβων στο DOM

- Εκτός της *διαπέρασης* του DOM tree για την αναζήτηση στοιχείων, μπορεί να γίνει και αναζήτηση βάσει:
  - του τύπου του html στοιχείου:
    Η `getElementsByTagName()` είναι μια **συνάρτηση** που επιστρέφει μια λίστα από τους απογόνους ενός κόμβου που είναι στοιχεία συγκεκριμένου τύπου
  - της κλάσης του html στοιχείου:
    Η `getElementsByClassName()` είναι μια **συνάρτηση** που επιστρέφει μια λίστα από τους απογόνους ενός κόμβου που είναι στοιχεία τα οποία δηλώνουν ότι ανήκουν σε μια συγκεκριμένη κλάση
  - του `id` ενός συγκεκριμένου στοιχείου:
    Η `getElementById()` είναι μια **συνάρτηση** που επιστρέφει **ένα** (ή `null`) απόγονο ενός κόμβου που έχει το συγκεκριμένο `id`

Πλήρης λίστα ιδιοτήτων και μεθόδων: https://developer.mozilla.org/en-US/docs/Web/API/Document

Πηγές

- Marijn Haverbeke, Eloquent JavaScript, 3rd edition (2018), 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