ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Εισαγωγή στον Προγραμματισμό με JavaScript

Αθανάσιος Ανδρούτσος

Εισαγωγή στην JavaScript

- Η JavaScript είναι η γλώσσα προγραμματισμού στο Web. Όλοι οι Web Browsers σε desktops, κινητά, tablets όλων των λειτουργικών συστημάτων υλοποιούν JavaScript Interpreters
- Εκτός από τους **Web Browsers**, από το 2010, η JavaScript με το **Node.js** χρησιμοποιείται και στο Back-end
- Το συντακτικό της γλώσσας μοιάζει σε γενικές γραμμές με το συντακτικό της Java, αλλά κατά τα άλλα οι δύο γλώσσες διαφέρουν μεταξύ τους

Core JavaScript Language

- Η JavaScript ορίζει ένα **minimal API** για να μπορούμε να δουλέψουμε κυρίως με το **Type System** που παρέχει, όπως numbers, text, booleans, arrays, objects, sets, maps κλπ.
- Αλλά δεν παρέχει input/output, ούτε networking, storage και graphics. **Αυτά βρίσκονται στην ευθύνη του host environment**
- Όπως αναφέραμε, δύο είναι τα περιβάλλοντα μεταγλώττισης και εκτέλεσης JavaScript: 1) **Web Browsers** στο front-end, και 2) **Node.js** στο back-end

Web Browsers Environment

- Το **περιβάλλον μεταγλώττισης** προγραμμάτων JavaScript στο front-end είναι οι **Web Browsers**
- Οι Web Browsers λειτουργούν ως *sandbox*. Για λόγους ασφαλείας δεν επιτρέπουν σε JavaScript προγράμματα να έχουν πρόσβαση στα αρχεία του Λειτουργικού Συστήματος. Παρέχουν ωστόσο ένα **local storage**, μία περιοχή αποθήκευσης στον Browser, όπου μπορεί η JavaScript να αποθηκεύει πληροφορίες
- Το **περιβάλλον του Browser επιτρέπει στην JavaScript να αλληλεπιδρά με φόρμες (<form>)**, όπως να διαβάζει από textboxes και άλλα controls, να εμφανίζει το output με HTML, να ανταποκρίνεται σε events και να στέλνει και να λαμβάνει HTTP requests/responses

Node.js

- Από το 2010 το Node.js (multiplatform **JavaScript interpreter & runtime**) έδωσε τη δυνατότητα ανάπτυξης JavaScript προγραμμάτων στο back-end χωρίς την ανάγκη ύπαρξης ενός Web Browser, και χωρίς τους περιορισμούς του API του Web Browser **προσφέροντας άμεση πρόσβαση στο Λειτουργικό Σύστημα** (read/write files, send/receive data με HTTP requests/response) όπως δηλαδή μία κανονική γλώσσα προγραμματισμού

JavaScript

- Η JavaScript είναι λοιπόν μία γλώσσα υψηλού επιπέδου και μαζί με τις γλώσσες HTML και CSS αποτελούν τις βασικές τεχνολογίες του World Wide Web
- Η JavaScript επιτρέπει **αλληλεπίδραση** του χρήστη με τις web εφαρμογές. Υπάρχουν πολλές έτοιμες βιβλιοθήκες συναρτήσεων

Μεταγλώττιση

- Η JavaScript είναι μία γλώσσα **διερμηνευόμενη (interpreted)** με μεταγλώττιση δύο βημάτων (two-step compilation)
- Στο 1ο βήμα δημιουργείται ενδιάμεσος κώδικας (δεν αποθηκεύεται σε ενδιάμεσο αρχείο, όπως τα .class της Java) και στη συνέχεια άμεσα ο ενδιάμεσος κώδικας εκτελείται με JIT (Just In-time Compilation, Jitter) που δίνει εξαιρετικό performance
- Η πιο γνωστή υλοποίηση του JavaScript interpreter και runtime στους browsers είναι το **V8 JavaScript Engine της Google**, που χρησιμοποιείται στον Chrome, Edge και Node.js

Πλεονεκτήματα

- Μεταφερσιμότητα (portability). Ένα πρόγραμμα JavaScript μπορεί να λειτουργήσει σε όλους τους browsers και όλα τα Λειτουργικά Συστήματα
- Είναι interpreted αλλά υποστηρίζει Just-In-Time (JIT) Compilation και είναι αποτελεσματική (efficient)
- Μπορούμε να γράψουμε συνοπτικό και αποτελεσματικό κώδικα

Χαρακτηριστικά

- **Dynamic typing.** Οι **δηλώσεις μεταβλητών** δεν περιλαμβάνουν τον τύπο δεδομένων της μεταβλητής, άρα **δεν μπορεί να γίνει έλεγχος τύπων και παραστάσεων @compile-time**. Ο τύπος δεδομένων μπορεί να αλλάζει δυναμικά @runtime ανάλογα με τα περιεχόμενα της μεταβλητής
- **Weakly-typed.** Η JavaScript ανάλογα το context προσπαθεί να κάνει αυτόματα conversions (coercion) ενός τύπου δεδομένων σε άλλον, π.χ. Number σε string, ή string σε boolean, κλπ.

JavaScript Ιστορική Διαδρομή

- Η JavaScript δημιουργήθηκε από την εταιρεία Netscape (τώρα Mozilla) τα πρώτα χρόνια ανάπτυξης του Web (1994 -) ενώ το όνομα 'JavaScript' είναι trademark της εταιρείας Sun Microsystems (σήμερα Oracle) για να περιγράψει την υλοποίηση της γλώσσας από την Netscape

ECMA

- Η Netscape υπέβαλε τη γλώσσα προς έγκριση στον οργανισμό ECMA (European Computer Manufacturer's Association) και το πρότυπο -λόγω του trademark από την Oracle- ονομάστηκε **ECMAScript**

JavaScript Versions

- Η έκδοση 5 της ECMAScript (2010) ήταν η βασική έκδοση της γλώσσας
- Το **2015 η ES6** προσέθεσε σημαντικά χαρακτηριστικά στη γλώσσα κυρίως όσο αφορά το Type System της γλώσσας, όπως Classes, Modules, Promises, Template Literals, Arrow Functions, Δηλώσεις Let και Const, Default παραμέτρους, Generators, Destructuring Assignment, Rest & Spread operator, Map/Set, κλπ., μετατρέποντάς τη σε γλώσσα γενικού σκοπού

ES / JS Versions

- Μετά την ES6, το 2015, η ECMAScript έχει περάσει σε ετήσια release του specification της γλώσσας – ES2016 (version ES7), ES2017 (v. ES8), ES2018 (v. ES9), ES2019 (v. ES10), ES2020 (v. ES11), ES2021 (v. ES12), ES2022 (v. ES13), ES2023 (v. ES14), ES2024 (v. ES15) και οι εκδόσεις της γλώσσας αναγνωρίζονται πλέον από το year of release

ECMAScript

- Η **JavaScript** (συντομογραφία JS) επομένως είναι μία γλώσσα γενικού σκοπού που βασίζεται στο πρότυπο *ECMAScript*
- Η πιο πρόσφατη έκδοση της ECMAScript είναι η ES15 (2024), ωστόσο όλοι οι browsers (Chrome, Firefox, Safari, Microsoft Edge) **υποστηρίζουν 100% την έκδοση ES6 (2015) και κάποια features των μεγαλύτερων εκδόσεων**

Type System (Primitives)

- Η JavaScript παρέχει επτά πρωταρχικούς τύπους δεδομένων (primitives). Τα **primitives στην JavaScript είναι immutable**:
  - **Number.** IEEE754 (1 πρόσημο + 52 mantissa + 11 exponent). Υποστηρίζει και ints και floating points
  - **String.** Αλφαριθμητικά, UTF-16. Ένα char είναι απλά string με length = 1
  - **Boolean.** True/false
  - **BigInt.** ES 2020, υποστηρίζει μεγάλους ακέραιους
  - **Undefined**. Default τιμή όταν δεν αρχικοποιούμε μία primitive μεταβλητή και τύπος δεδομένων
  - **Null**. Default τιμή όταν δεν αρχικοποιούμε ένα object και τύπος δεδομένων
  - **Symbol**. Λειτουργεί ως τύπος δεδομένων που δίνει UIDs (Unique IDs)

Primitives

Type	`typeof` return value	Object wrapper
Null	`"object"`	N/A
Undefined	`"undefined"`	N/A
Boolean	`"boolean"`	`Boolean`
Number	`"number"`	`Number`
BigInt	`"bigint"`	`BigInt`
String	`"string"`	`String`
Symbol	`"symbol"`	`Symbol`

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

Σύνθετοι Τύποι Δεδομένων

- **Arrays**. Στην JavaScript τα arrays είναι zero-based, δυναμικές (grow/shrink αυτόματα) ακολουθίες στοιχείων οποιουδήποτε τύπου
- **Objects**. Πρόκειται για δομή που περιέχει **properties** στη μορφή **key: value**. Το value μπορεί να είναι data type ή function (τα functions δηλώνονται ως properties μέσα στο object). Τεχνικά, **πρόκειται για instances του Object type**, που είναι ο root type στην ιεραρχία κληρονομικότητα της JavaScript
- Τα arrays και objects είναι structurally modifiable (μπορούμε να κάνουμε add, modify, remove elements). Μερικές φορές αυτό αναφέρεται και ως mutable

Identifiers

- Τα identifiers στην JavaScript είναι **case sensitive**, ξεκινούν με **γράμμα, _, $** και στη συνέχεια περιλαμβάνουν γράμματα, _, $ και ψηφία
- Δηλώσεις μεταβλητών πρωταρχικών τύπων καθώς και objects ή sets/maps κάνουμε με **var, let, const**
- Δείτε στη συνέχεια τα keywords var, let, const και τη λειτουργία τους

Δηλώσεις μεταβλητών

- Δηλώσεις μεταβλητών μπορούμε να κάνουμε
  - Σε συναρτήσεις (**local scope**)
  - Σε δομές ελέγχου if, for, while (**block-scope**)
  - Σε modules, δηλαδή αρχεία JavaScript που λειτουργούν ως βιβλιοθήκες διαμοιραζόμενων συναρτήσεων και μεταβλητών (**module scope**)
  - Έξω από όλες τις συναρτήσεις οπότε τότε το scope είναι το **global scope** (σε script mode, όχι modules)

Δηλώσεις

- **Var**. π.χ. var num = 5; Οι δηλώσεις var **έχουν προβλήματα** και δεν προτείνονται ιδιαίτερα μετά την ES6 όπου ήρθαν στη γλώσσα οι δηλώσεις let και const
  - Μπορούν να δηλωθούν στο function scope, module scope ή global scope, αλλά **όχι block scope**
  - **Hoisting**. Μπορούν να χρησιμοποιηθούν πριν δηλωθούν, οπότε τότε η τιμή τους είναι undefined
  - Όταν δηλώνονται στο global scope, τότε γίνονται **properties του global object** (ειδικό object της JS, με όνομα **window** στον browser και **global** στο Node.js), οπότε και μπορεί να δημιουργηθούν naming conflicts
  - **Redeclaration**. π.χ. var x = 10; var x = 20;

Let και const

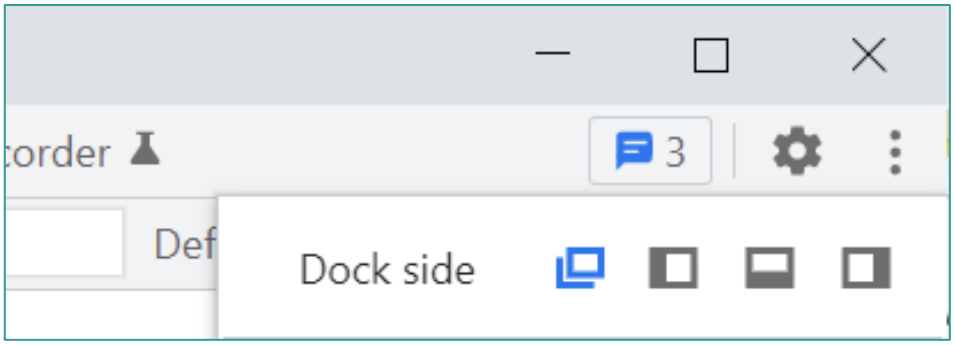
- Από την ES6 και μετά, μπορούμε να κάνουμε δηλώσεις let και const, π.χ. let num = 0; const PI = 3.14; const arr = [1, 2, 3]. Αν δηλώσουμε χωρίς αρχικοποίηση, η default τιμή είναι το undefined
- Μπορούμε να δηλώνουμε επίσης, στο block scope
- Γίνονται hoisted αλλά δεν μπορούν να χρησιμοποιηθούν πριν αρχικοποιηθούν (temporal dead zone)

Stack / Heap

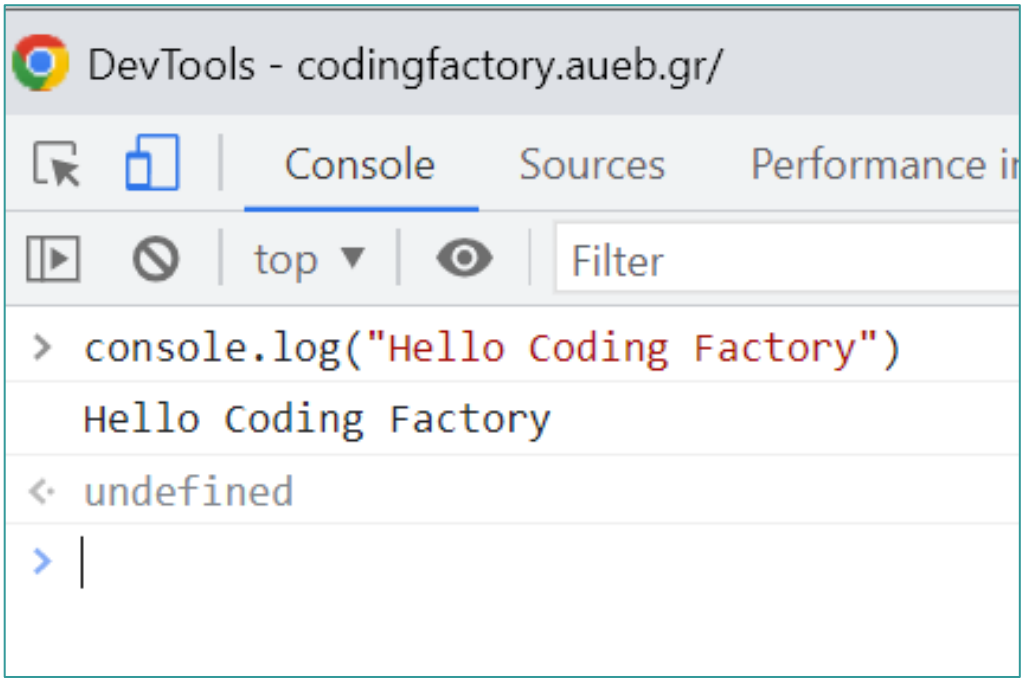
- Στο V8 της Google όλα τα primitives (εκτός από τα Smis – small integers) αποθηκεύονται στο heap. Στο stack είναι οι δείκτες (references)
- Το ίδιο βέβαια ισχύει και για τα objects.
- Υπάρχει garbage collector
- Τα strings γίνονται intern (interning pool)

Περιβάλλον Web Browser

- **Web Developer Tools**: Με ***Ctrl-Shift-I*** ή ***Command-Option-I*** ή ***F12*** και επιλέγουμε Console
- Τα **Web Developer Tools** συνήθως εμφανίζονται σε pane κάτω ή δεξιά του browser window αλλά μπορούμε να κάνουμε και detach σε ξεχωριστό παράθυρο



Πρώτο πρόγραμμα

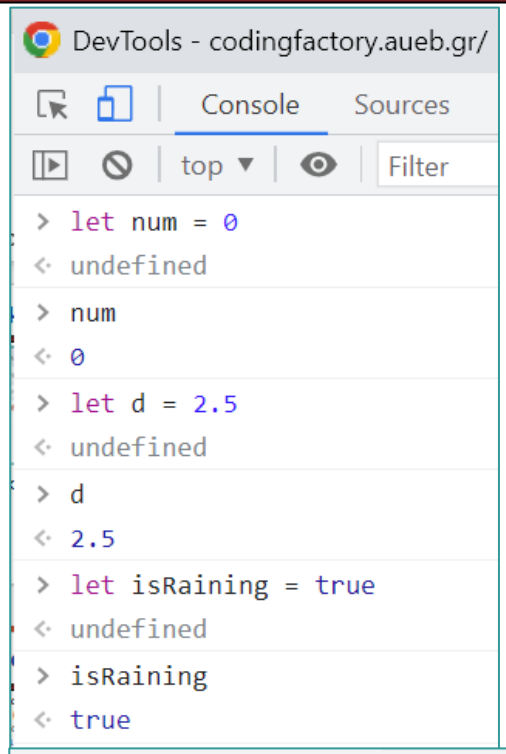


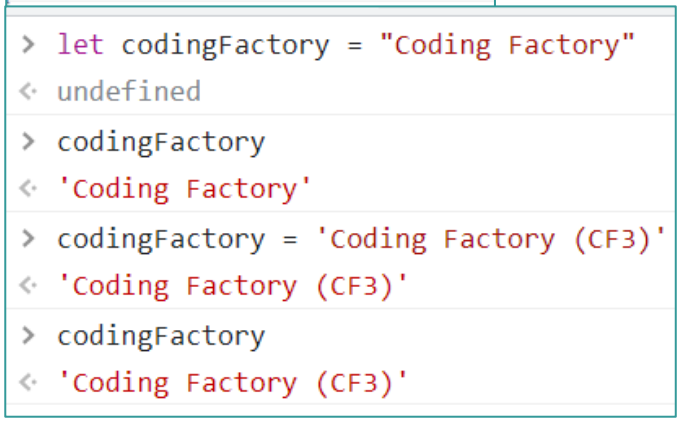
- Με console.log() εκτυπώνουμε στην κονσόλα

Exploring JavaScript

- Στη συνέχεια θα δούμε όλα τα προηγούμενα πιο αναλυτικά μέσα από παραδείγματα
- Θα χρησιμοποιήσουμε την κονσόλα του browser

Exploring JavaScript

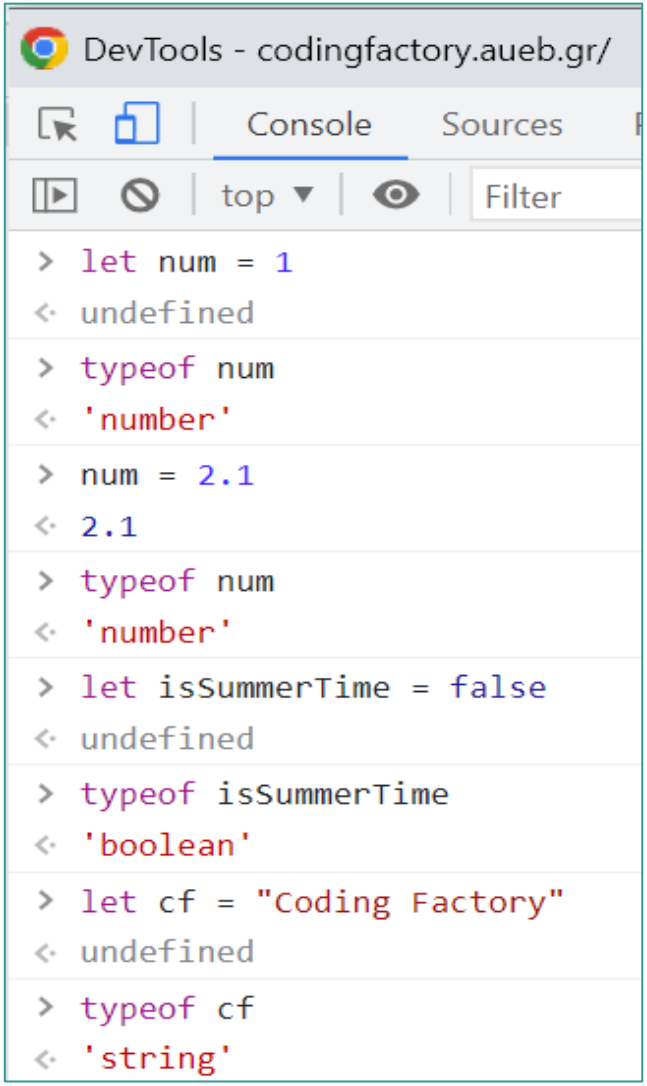




- Με **let δηλώνουμε μεταβλητές**. Ο τύπος τους γίνεται infer (συμπεραίνεται) από την τιμή. Στο περιβάλλον του browser, δίνοντας το όνομα της μεταβλητής υπολογίζεται και εμφανίζεται η τιμή
- **Numbers** συνήθως δηλώνουμε και αρχικοποιούμε με literals. (0, 5, 7.14, κλπ.) Δεν υπάρχει ξεχωριστός τύπος int, υπάρχει BigInt (τα BigInt literals ορίζονται με n στο τέλος, π.χ. 900112n).
- Booleans δηλώνουμε και ορίζουμε με boolean literals (true / false)

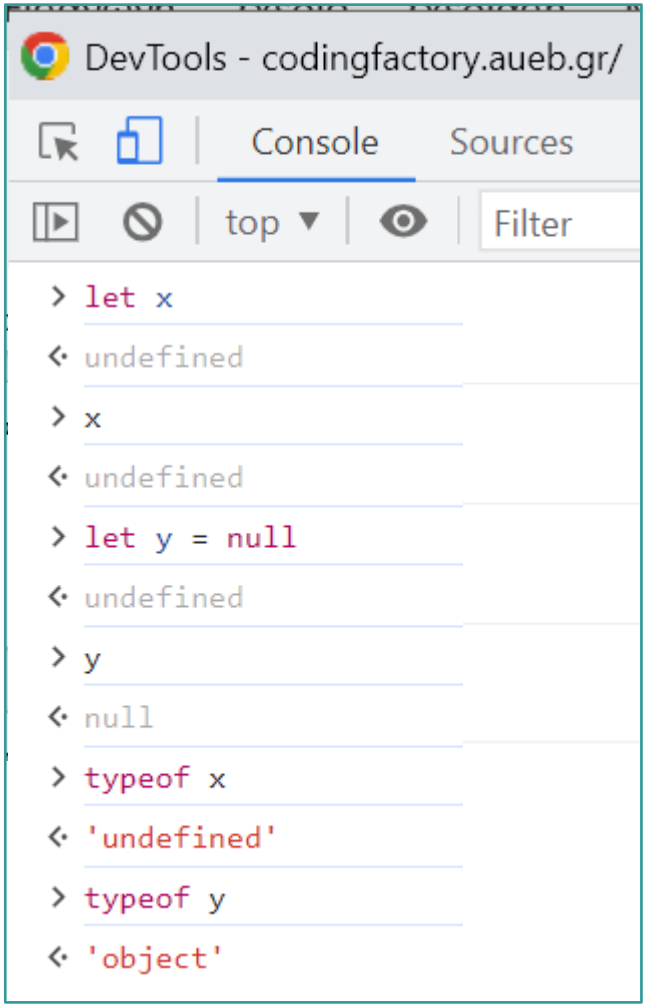
- **String** literals μπορούμε να δώσουμε είτε με single quotes ' ' ή με double quotes " " ή με backticks ` `
- Αν η έκφραση δεν επιστρέφει μια τιμή, η κονσόλα θα αναφέρει *undefined* — αυτό δεν σημαίνει ότι κάτι δεν πάει καλά, απλώς η κονσόλα του Chrome/Firefox ενημερώνει ότι η προηγούμενη έκφραση δεν επέστρεψε αποτέλεσμα

Primitives



- Τα number, boolean και string θεωρούνται primitives (Πρωταρχικοί τύποι δεδομένων)
- Τα primitives είναι immutable
- Η JavaScript –δεδομένου ότι υλοποιεί dynamic policy- τα υλοποιεί διατηρώντας ένα δείκτη στο stack και το περιεχόμενο στο heap
- **Το περιεχόμενο στο heap περιλαμβάνει την τιμή αλλά και τον τύπο του αντικειμένου**, ώστε να μπορεί να κάνει dynamic type checking

Null & Undefined



- Τα *null* και *undefined* θεωρούνται primitives
- Το ***undefined*** είναι όταν έχουμε δηλώσει μία μεταβλητή αλλά δεν έχουμε αρχικοποιήσει
- Το undefined είναι τιμή.
- Το **null** είναι τιμή για απουσία τιμής και χρησιμοποιείται σε reference types. Πρέπει ρητά να δηλώσουμε π.χ. let arr = null;

Σύνθετοι Τύποι - Objects

DevTools - codingfactory.aueb.gr/

Console | Sources

top | Filter

```
> const user = {
      id: 1,
      firstname: "Alice",
      lastnane: "W."
  }
<- undefined
> user.firstname
<- 'Alice'
> user['lastname']
<- 'W.'
> user.age = 20
<- 20
> user.hobbies = []
<- ▶ []
> typeof user
<- 'object'
```

- Τα **objects είναι συλλογές από key/value pairs** όπως τα πεδία των instances μίας κλάσης με ένα συγκεκριμένο state
- Τα **keys** είναι στην πραγματικότητα (μετατρέπονται σε) strings. Τα **values** μπορεί να είναι primitives ή σύνθετοι τύποι
- Τα objects τεχνικά είναι **instances του τύπου *Object*** που είναι υπερτύπος όλων των κλάσεων/τύπων της JavaScript
- Εδώ δηλώνουμε και αρχικοποιούμε με **object literal / initializer**. Μπορούμε και **προσθέτουμε πεδία** όπως το age ή hobbies (είναι πίνακας)

Πρόσβαση στα object properties


```
> user['id']
< 1
> user["firstname"]
< 'Alice'
```

```
> let first = "firstname"
< undefined
> user[first]
< 'Alice'
```

- Πρόσβαση στα πεδία του object έχουμε είτε με τον **τελεστή τελεία (.) ή με τον τελεστή []** που παίρνει string ως indexing, π.χ. user[“id“] (ή user[‘id’])
- Στην πρόσβαση στα πεδία ενός object με τον τελεστή [], το πλεονέκτημα είναι ότι **μπορούμε να περάσουμε ως παράμετρο και μεταβλητή string** ενώ στην σύνταξη με τελεία, δεν μπορούμε

Object Constructor


```
> const bob = new Object()
<- undefined
> bob.id = 2
<- 2
> bob.firstname = "Bob"
<- 'Bob'
> bob.lastname = "M."
<- 'M.'
> bob.age = 21
<- 21
```

- Δημιουργία object μέσω του Constructor
- Στη συνέχεια προσθέτουμε ιδιότητες

- Ωστόσο, συνήθως –όπως αναφέραμε- αρχικοποιούμε με object literal

for .. in


```
> for (const property in user) {
      console.log(property, user[property])
  }
```

```
id 1
firstname Alice
lastnane S.
hobbies ▶ []
```

- Η for .. in διατρέχει μόνο τα enumerated strings ενός object
- Παρατηρούμε πως μέσα σε backticks `` μπορούμε να ενθέσουμε μεταβλητές με ${}. Αυτό ονομάζεται **interpolation**

for .. of


```
> for (const [key, value] of Object.entries(user))
      console.log(`${key} : ${value}`)
  }
  id : 1
  firstname : Alice
  lastname : W.
  age : 20
  hobbies : [object Object]
```

- Η **for .. of** διατρέχει γενικά Iterable objects όπως πίνακες, strings, sets, maps και γενικά collections που είναι Iterable
- Η static μέθοδος **Object.entries()** επιστρέφει ένα πίνακα που τα στοιχεία του (entries) είναι πίνακες με δύο στοιχεία, και μπορούμε και κάνουμε destucturing assignment στα [key, value] οπότε και κάνουμε traverse με for .. Of
- Destructuring assignment σημαίνει ότι εκχωρούμε τα στοιχεία του πίνακα σε μεταβλητές τις οποίες δηλώνουμε μέσα σε [ ] στην αριστερή πλευρά της δήλωσης

Template Literals


```
> for (const [key, value] of Object.entries(user)) {
      console.log(`${key} : ${value}`)
  }
```

- Παρατηρούμε επίσης όπως είπαμε ότι η console.log() παίρνει ως παράμετρο ένα template literal (template string) μέσα σε backticks. Μέσα σε template literals μπορούμε και κάνουμε interpolate μεταβλητές που είναι μια πιο advanced μορφή των placeholders της Java printf
- Τα interpolated vars τα εισάγουμε με τη σύνταξη ${} δηλαδή δολάριο ακολουθούμενο από μονά μουστάκια (curly brackets/mustache syntax). Μέσα στα curly brackets εισάγουμε τη μεταβλητή ή το expression που θέλουμε να εμφανίσουμε

Traverse bob object


```
> for (const [key, value] of Object.entries(bob)) {
      console.log(`${key} : ${value}`)
  }
  id : 2
  firstname : Bob
  lastname : M.
  age : 21
  address : [object Object]
< undefined
> for (const [key, value] of Object.entries(bob.address)) {
      console.log(`${key} : ${value}`)
  }
  street : Patission
  number : 71
  zipcode : 10434
```

- Με for .. of κάνουμε traverse το bob καθώς και το *bob.address* που είναι επίσης ένα object μέσα στον bob

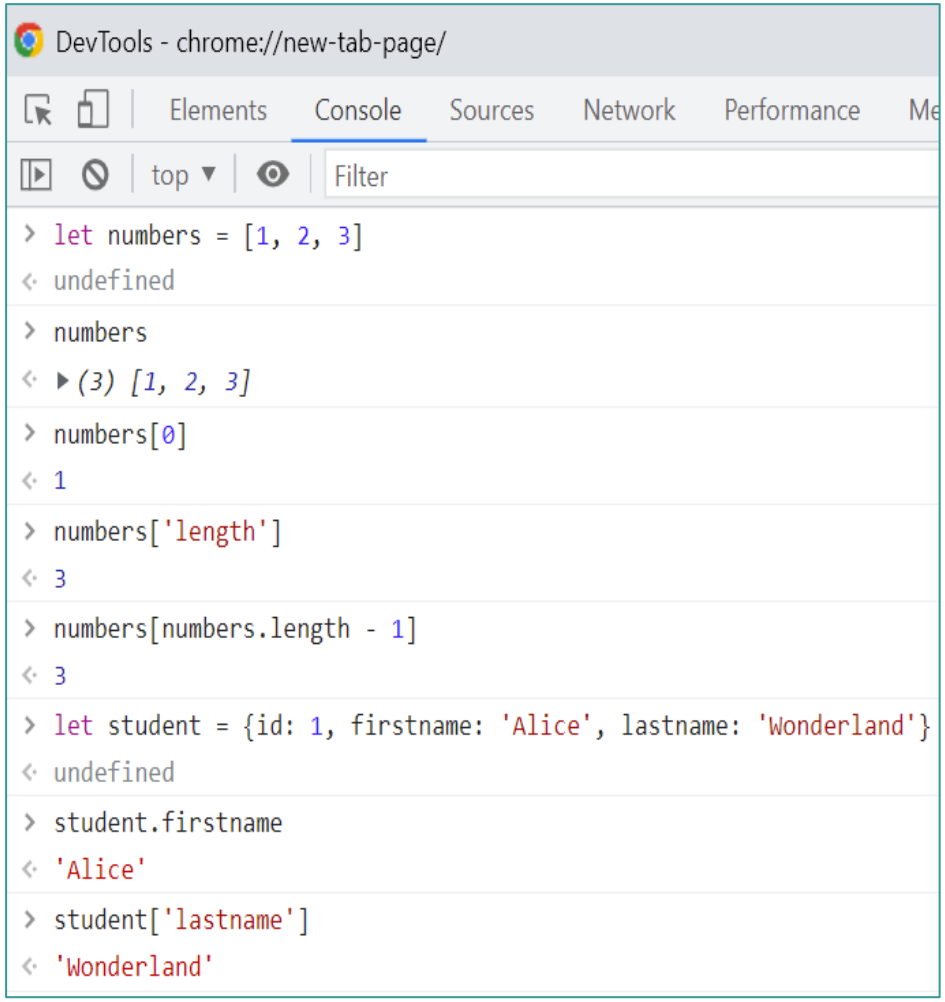
Arrays


```
> const cities = ["Athens", "Paris", "London", "Berlin"]
<- undefined
> cities.length
<- 4
> cities[0]
<- 'Athens'
> cities["0"]
<- 'Athens'
> cities[cities.length - 1]
<- 'Berlin'
```

- **Arrays ορίζουμε μέσα σε [].** Μπορούν να περιέχουν τιμές όλων των τύπων δεδομένων, primitives ή object ή συνδυασμούς αυτών
- Αυτό δεν σημαίνει ότι είναι καλή πρακτική να έχουμε διάφορους τύπους δεδομένων μέσα σε ένα πίνακα

- Τα Arrays στην JavaScript υλοποιούνται ως (special) objects. Τα indexes του πίνακα γίνονται properties του object ενώ προστίθεται στο object και η ιδιότητα length. Αφού τα indexes γίνονται properties, μπορούμε να αναφερθούμε και ως cities["0"] (βλ. παράδειγμα) στη θέση 0 του πίνακα, μιας και τα keys των objects παρότι μπορεί να δηλώνονται ως numbers, υλοποιούνται τελικά ως strings. Το indexing των πινάκων είναι από 0 έως length - 1

Arrays



- Πίνακες αρχικοποιούμε με []
- Οι πίνακες στην JavaScript αυξάνουν το μέγεθός τους αυτόματα (δυναμικά, δεν είναι στατικοί όπως στην Java)
- Το index των πινάκων ξεκινά από το 0 και αναφερόμαστε στις θέσεις πινάκων με arr[index] δηλ. arr[0] ή arr[1] κλπ.

For .. of σε arrays


```
> for (const city of cities) {
      console.log(city)
  }
  Athens
  Paris
  London
  Berlin
```

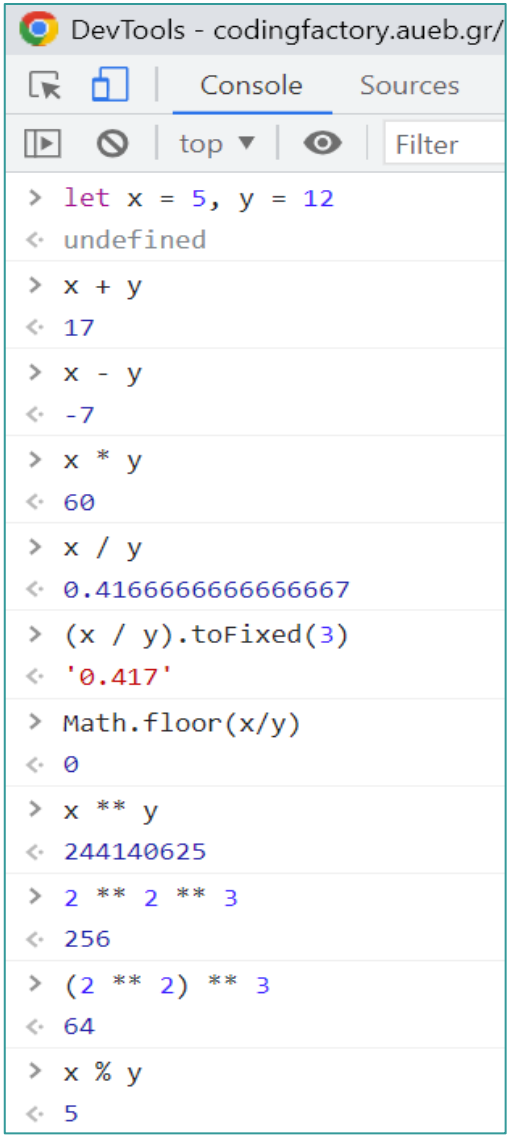
- Με **for .. of** μπορούμε και διατρέχουμε τα στοιχεία ενός array, αφού τα arrays είναι Iterable

forEach


```
> const arr = [1, 2, 3, 4, 5]
< undefined
> arr.forEach(num => console.log(num))
  1
  2
  3
  4
  5
```

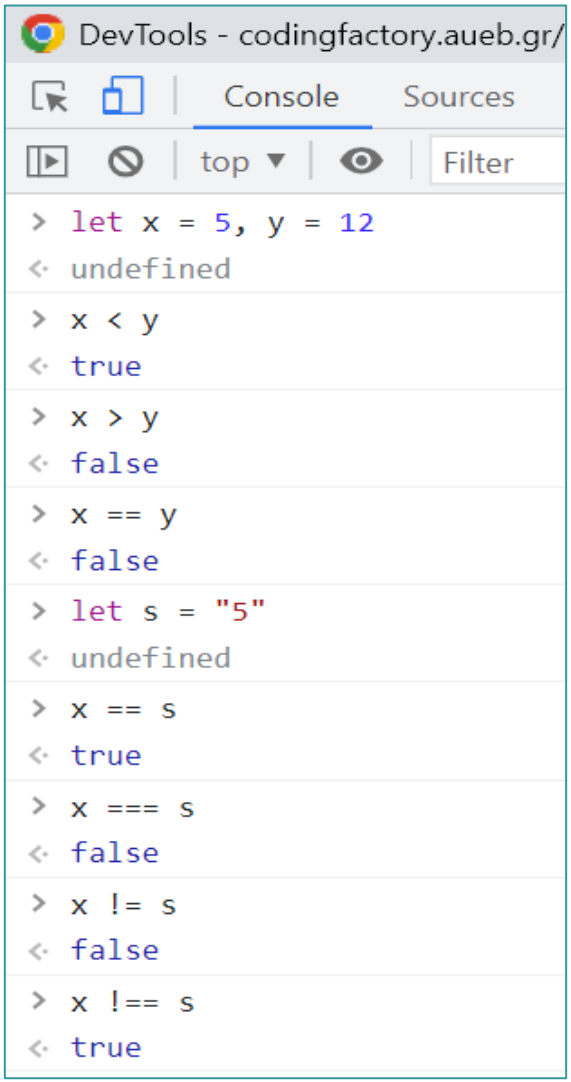
- Η forEach είναι μία έτοιμη μέθοδος των iterables όπως τα arrays. Λαμβάνει ως είσοδο ένα **callback**, μία συνάρτηση δηλαδή που μπορούμε και δίνουμε και σε μορφή arrow function
- Θα αναφερθούνε σε callbacks στα επόμενα

Αριθμητικοί Τελεστές



- Στην JavaScript η διαίρεση ακεραίων δίνει δεκαδικό
- Για να πάρουμε το ακέραιο μέρος μπορούμε με *Math.floor()*
- Με *.toFixed(n)* παίρνουμε η πλήθος δεκαδικών ψηφίων
- Παρατηρούμε ότι ενώ το x/y είναι number άρα primitive μπορούμε και καλούμε methods
- **Αυτό είναι δυνατό επειδή η JavaScript κάνει αυτόματα wrap στους wrapper τύπους των primitives (Number, Boolean, String)**
- Η ύψωση σε δύναμη γίνεται με τον τελεστή ** ο οποίος έχει δεξί associativity 2 ** 2 ** 3 είναι 2 ** 8 και όχι 4 ** 3
- Με παρενθέσεις ορίζουμε προτεραιότητες.
- Modulo παίρνουμε με το %

Τελεστές Σύγκρισης



- Στη JavaScript ισότητα **ελέγχουμε με === και ανισότητα με !==**
- Τα == και != δεν χρησιμοποιούνται συχνά πλέον μιας και δίνουν ισότητες σε στοιχεία διαφορετικών τύπων. Για παράδειγμα, όταν ο ένας τύπος μπορεί να γίνει coerce σε ένα άλλο (π.χ. 5 == "5") είναι true, ενώ 5 === "5" είναι false

Λογικοί Τελεστές (1)

DevTools - codingfactory.aueb.gr/

Console | Sources | Performance insights

top | Filter

```
> let isAdult = true
undefined
> let age = 30
undefined
> let isEligible = isAdult && (age > 50)
undefined
> isEligible
false
> isEligible = isAdult || ((age > 12) && (age <= 18))
true
> !isEligible
false
```

- && (AND), || (OR), ! (NOT). Είναι short-circuit.
- **Επιστρέφουν έναν από τους τελεσταίους**
- && -- τον πρώτο falsy ή τον τελευταίο truthy αν όλοι οι τελεστές είναι true
- || -- τον πρώτο truthy ή τον τελευταίο falsy αν όλοι οι τελεστές είναι false
- ! -- True αν ο τελεσταίος είναι falsy, false αν ο τελεσταίος είναι true

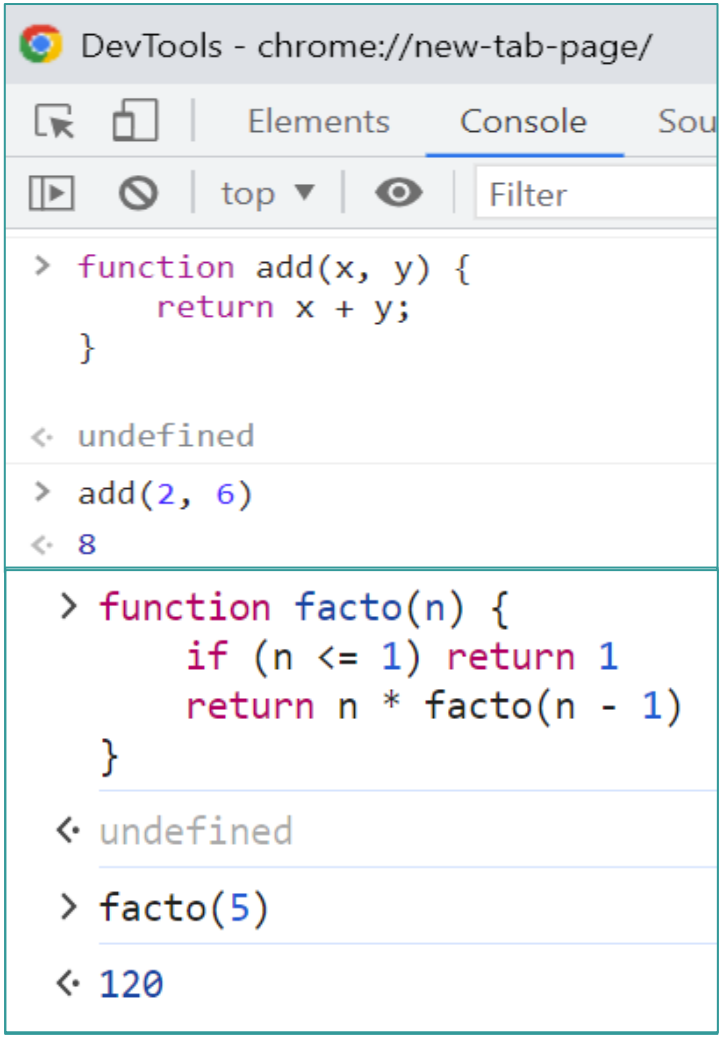
```
> let a = 5
undefined
> let res = a || 0
undefined
> res
5
```

```
> let b;  // undefined
undefined
> let res2 = b || 0
undefined
> res2
0
```

Λογικοί Τελεστές (2)

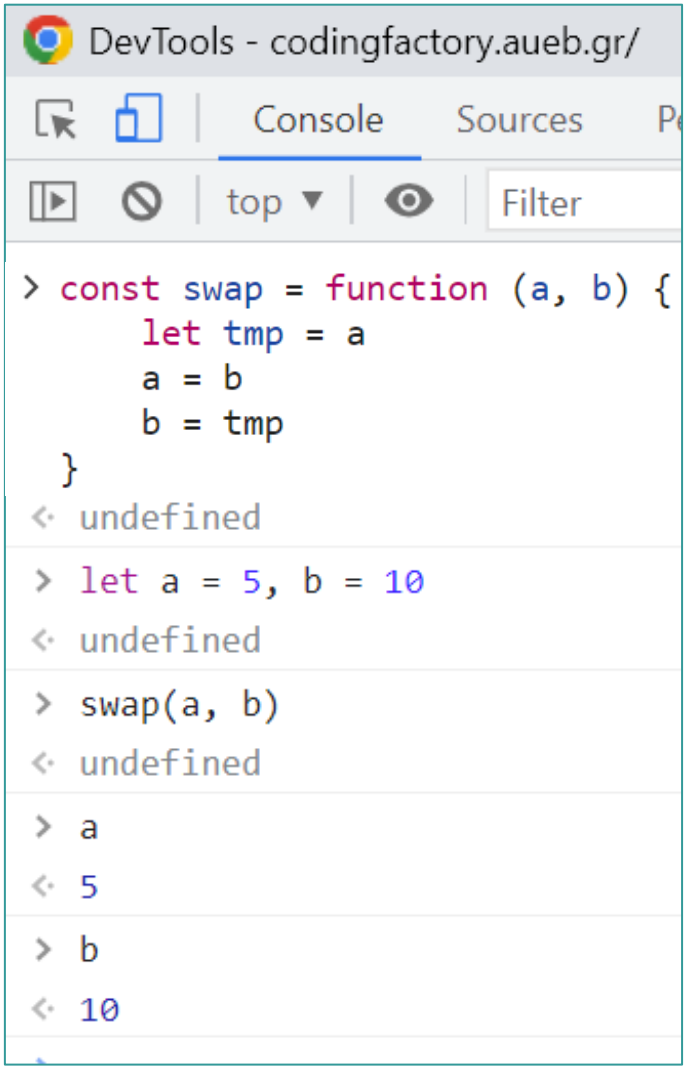
- Για παράδειγμα στην εκχώρηση let user = username || "default user"
- Αν το username είναι undefined ή null ή κενό string, τότε θα εκχωρηθεί στο user, το "default user"
- Αλλιώς θα εκχωρηθεί το username

Συναρτήσεις



- Συναρτήσεις στην JavaScript μπορούμε και ορίζουμε και ανεξάρτητα και όχι μόνο μέσα σε κλάσεις ως μεθόδους κλάσεων
- Μία συνάρτηση την ορίζουμε με το **keyword function**
- Παρατηρούμε πως **δεν έχουμε τύπους δεδομένων στις παραμέτρους** όπως στην Java αλλά μόνο τους identifiers χωρίς τύπους, **ούτε επιστρεφόμενες τιμές στην επικεφαλίδα**

Εκχώρηση συναρτήσεων



- Οι functions στην JavaScript έχουν **type function αλλά είναι special objects** που **μπορούν και εκχωρούνται** όπως στο παράδειγμα εκχωρείται στο swap μία συνάρτηση. Το όνομα της συνάρτησης είναι τώρα swap
- Επίσης, **οι παράμετροι περνάνε πάντα by value**

Arrow functions

- Από την **ES6 έχουμε και arrow functions**. Τα arrow functions είναι συναρτήσεις σε σύντμηση, όπου αριστερά από το fat arrow => είναι οι τυπικές παράμετροι και δεξιά το επιστρεφόμενο αποτέλεσμα
- Αν το αποτέλεσμα είναι απλά ένα expression που επιστρέφουμε δεν χρειάζεται άγκιστρα, ούτε return (υπονοείται). Αν χρησιμοποιήσουμε άγκιστρα, θέλει return
- Επίσης, αν στο δεξί μέρος δεν έχουμε ένα μόνο expression, χρειάζονται άγκιστρα και return, όπως σε ένα κανονικό πρόγραμμα

```
> const add = (a, b) => a + b
<· undefined
> add(3, 5)
<· 8
```

```
> const sub = (a, b) => {
      let dif = a - b
      return dif
  }
<· undefined
> sub(8, 2)
<· 6
```

Destructuring Assignment (1)


```
>> [a, b] = [1, 2]
<- ▶ Array [ 1, 2 ]
>> a
<- 1
>> b
<- 2
```

DevTools - codingfactory.aueb.gr/

Console | Sources | P

top | Filter

```
> let a = 5, b = 10
<- undefined
> [a, b] = [b, a]
<- ▶ (2) [10, 5]
> a
<- 10
> b
<- 5
```

- Unpack τις τιμές ενός array
- Μπορούμε εύκολα να κάνουμε swap με destructive assignment
- Στο destructive assignment στους πίνακες, ο πίνακας δεξιά εκχωρεί **ένα προς ένα τα στοιχεία του** στις μεταβλητές που βρίσκονται μέσα σε [] στα αριστερά
- Το [] στα αριστερά δεν σημαίνει πίνακας, είναι syntax για εκχώρηση ένα προς ένα των στοιχείων του πίνακα δεξιά σε μεταβλητές αριστερά

Destructuring Assignment (2)


```
> let obj = { first: "Athanassios", last: "Androutsos" }
< undefined
> let {first, last} = obj
< undefined
> first
< 'Athanassios'
> last
< 'Androutsos'
```

- **Destructive assignment μπορούμε να έχουμε και στα objects.** Κάνουμε unpack τις τιμές ενός object που βρίσκεται στα δεξιά μιας παράστασης και **εκχωρούμε τις ιδιότητες σε μεταβλητές με το ίδιο όνομα** όπως οι ιδιότητες στο αριστερό μέρος της παράστασης Τα ονόματα των μεταβλητών **first, last (πρέπει να) είναι ίδια, όπως αναφέραμε, με τις ιδιότητες του αντικειμένου**

```
>> let {first: firstname, last: lastname} = {first: "Athanassios", last: "Androutsos"}
← undefined
>> firstname
← "Athanassios"
```

- Μπορούμε ωστόσο να κάνουμε rename και να δώσουμε δικά μας identifiers με first: firstname ή last: lastname όπως στο παράδειγμα

Objects με μεθόδους

- Συναρτήσεις (μεθόδους) μπορούμε να έχουμε και στα objects, σαν ιδιότητες. Από την ES6 και μετά μπορούμε να ορίζουμε με concise method syntax

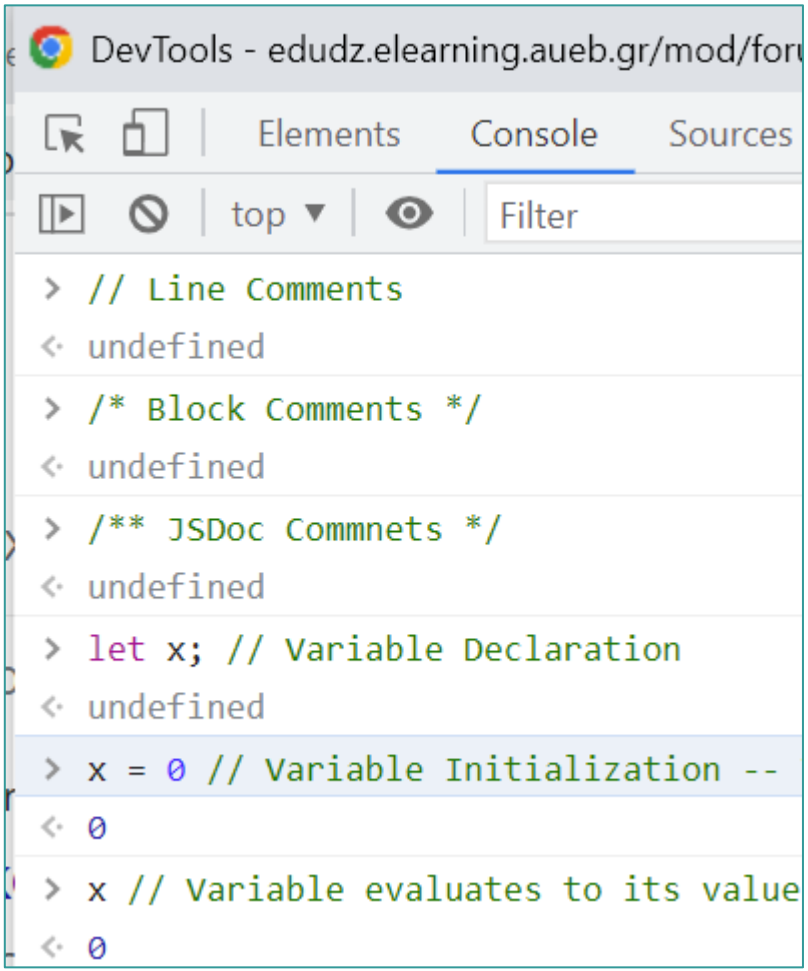
```
> const point = {
      x: 5,
      y: 10,
      printPoint: function () {
          console.log("(" + this.x + ", " + this.y + ")")
      }
  }
```

```
> const point = {
      x: 5,
      y: 10,
      printPoint() {
          console.log("(" + this.x + ", " + this.y + ")")
      }
  }
```

- Στις ιδιότητες του object, μέσα στη συνάρτηση αναφερόμαστε με this

```
> point.printPoint()
(5, 10)
```

Συνοπτικά η JavaScript (1)



- Βασικά θέματα σύνταξης όπως σχόλια, δηλώσεις μεταβλητών, αρχικοποιήσεις και απόδοση τιμής στην κονσόλα του Chrome

Συνοπτικά η JavaScript (2)


```
> let x = 5 // Declare and init a Number
< undefined
> let d = 10.5 // Number type in JavaScript includes integers and floats
< undefined
> let x = 123456789n // BigInt Type init witg n type-suffix
< undefined
> let x = BigInt(123456789) // BigInt Type
< undefined
> let s = "I am a String" // String Type
< undefined
> let s = 'I am a String' // String again in single quotes
< undefined
> let s = "Hello 'Coding Factory'" // Μπορούμε να έχουμε μονά εισαγωγικά μέσα σε διπλά
< undefined
> let s = 'Hello "Coding Factory"' // ή Διπλά μέσα σε μονά
< undefined
> let s = `Hello Again` // Back ticks define Template literals που μπορούν και κάνουν intrpolate μεταβλητές
< undefined
```

- Number, String, δύο πρωταρχικοί τύποι δεδομένων στην JS

Συνοπτικά η JavaScript (3)


```
> let isSunny = true
<· undefined
> let isRaining = false
<· undefined
> let x = null // null pointer
<· undefined
> let x = undefined // default τιμή όταν δεν αρχικοποιούμε
```

- Boolean, null και undefined επίσης πρωταρχικοί τύποι δεδομένων

Συνοπτικά η JavaScript (3)


```
> let arr = [1, 2, 3, 4, 5] // Array Data Type -- Arrays are dynamic in JS
< undefined
> arr[0] // Index of arrys start at 0 and we refer with arr[index]
< 1
> arr[arr.length - 1] // The last element of the array
< 5
> arr.length // The length ofthe array is given bu length property
< 5
> let array2d = [[1, 2], [3, 4]] // Δισδιάστατος array - The elements of the array are arrays
< undefined
>
```

- Πίνακες, μονοδιάστατοι και δισδιάστατοι με ταυτόχρονο initialization. Οι πίνακες έχουν δυναμικό size στην JS. Αυξάνουν και μειώνουν αυτόματα το μέγεθός τους

Συνοπτικά η JavaScript (4)


```
> let cf = {
      name: "Coding Factory",
      version: "2.1",
      year: 2022
  }
<· undefined
> cf.name // Access with .
<· 'Coding Factory'
> cf['name'] // Access as associative array with []
<· 'Coding Factory'
> Object.values(cf) // all cf values with Object class
<· ▶ (3) ['Coding Factory', '2.1', 2022]
> Object.keys(cf) // All keys of cf with Object class
<· ▶ (3) ['name', 'version', 'year']
> Object.entries(cf)[1] // All entries with Object.entries and then get entry[1] -- 2nd entry
<· ▶ (2) ['version', '2.1']
```

- Αντικείμενα (objects) και αναφορές σε αυτά

Συνοπτικά η JavaScript (5)


```
> const PI = 3.14 // constants with const keyword
<- undefined
> console.log(PI)
  3.14
<- undefined
> console.log("Hello Coding Factory")
  Hello Coding Factory
<- undefined
> function sayHelloAgain() {
      console.log("Hello Again")
  }
<- undefined
> sayHelloAgain()
  Hello Again
<- undefined
```

- Σταθερές (με const) και console.log() για να γράφουμε στην κονσόλα
- Ορίζουμε συναρτήσεις με το keyword function

Συνοπτικά η JavaScript (6)


```
> 1 + 2
< 3
> (1 + 2) * ( 5 - 4)
< 3
> 5 / 4
< 1.25
> 5 % 4
< 1
> (5 / 4).toFixed(1)
< '1.3'
> (5 / 4).toExponential()
< '1.25e+0'
> Math.floor(5 / 4)
< 1
> Math.ceil(5 /4)
< 2
> Math.round(5 / 4)
< 1
> 2 ** 3 ** 2 // right associative
< 512
> (2 ** 3) ** 2 // Parentheses for left associativity
< 64
```

- Αριθμητικοί τελεστές
- Η διαίρεση ακεραίων δίνει δεκαδικό
- Το mod (%) δίνει το ακέραιο υπόλοιπο
- Συναρτήσεις και η κλάση Math μας δίνουν περαιτέρω λειτουργίες

Συνοπτικά η JavaScript (7)


```
> Number("123") // Typecast - Conversion String -> Number
< 123
> (123).toString() // Όπως πριν, conversion σε String
< '123'
> String(123)
< '123'
```

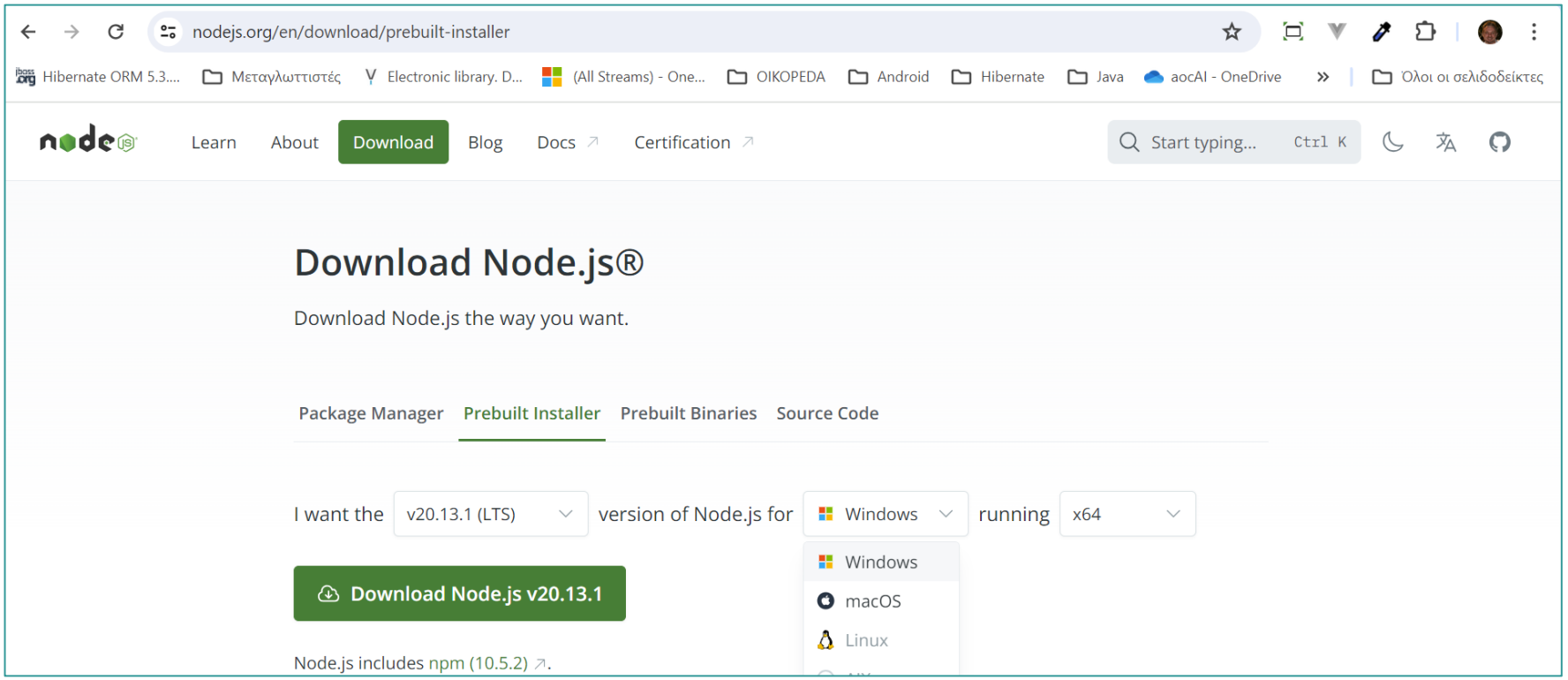
- Convert String – Number
- Οι wrapper κλάσεις Number και String μετατρέπουν διάφορους τύπους δεδομένων στον τύπο του αντικειμένου

Συνοπτικά η JavaScript (8)


```
> let i = 0;
undefined
> i++ // post-increment, access value will be 0
0
> ++i // pre-icrement, access va;ue will be 2
2
> i += 5
7
> let x = 2
undefined
> let y = 2
undefined
> x === y
true
> x = '2'
'2'
> x == y
true
> x === y
false
```

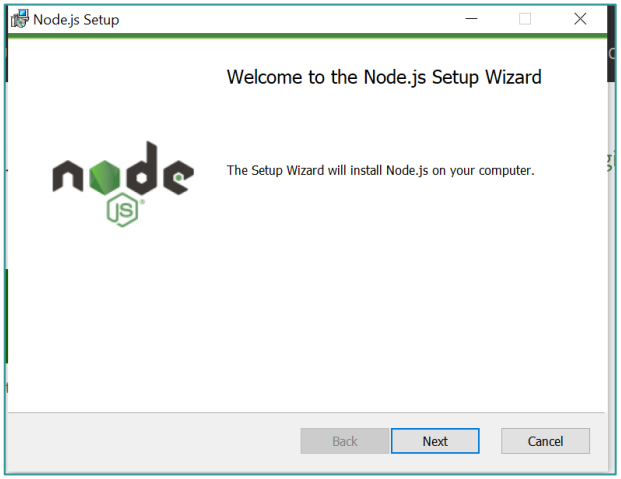
- Συντμήσεις αριθμητικών τελεστών
- Τελεστές σύγκρισης
- Συγκρίνουμε με === ώστε να ληφθεί υπόψη και ο τύπος δεδομένων
- Όχι με == όπου γίνονται αυτόματες μετατροπές τύπων και δίνονται ψευδείς ισότητες

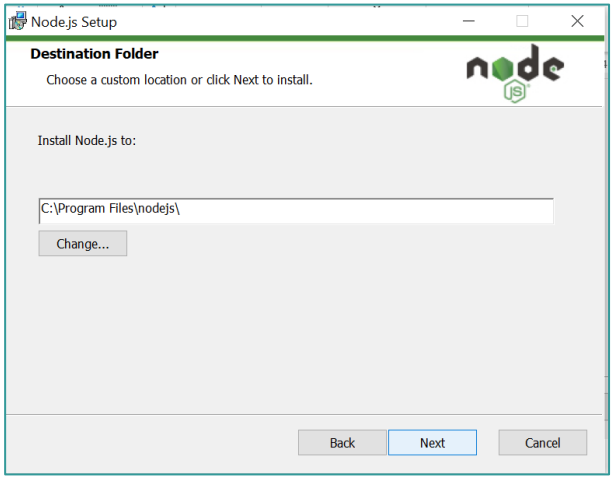
Node install (1)

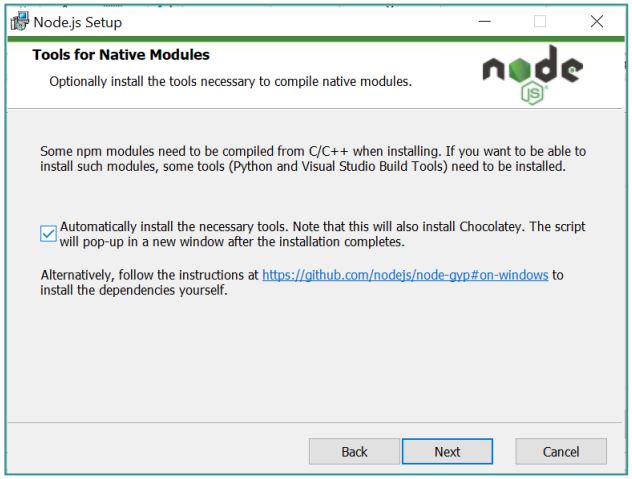


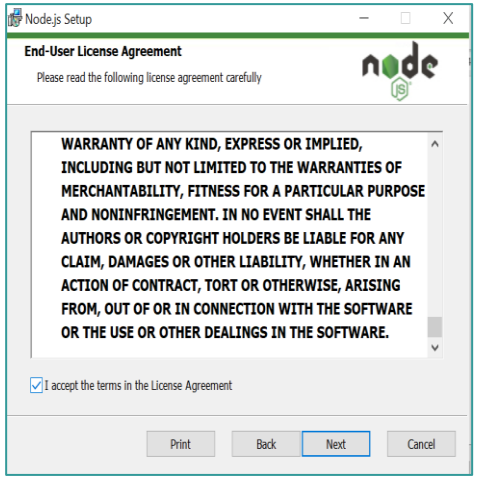
- Υπάρχει έκδοση για όλα τα Λειτουργικά Συστήματα
- https://nodejs.org/en/download/prebuilt-installer

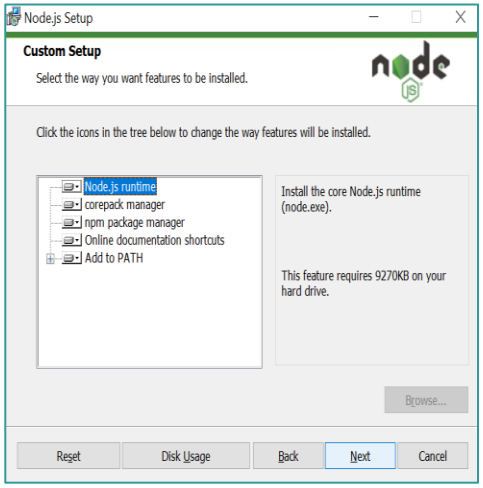
Node install (2)

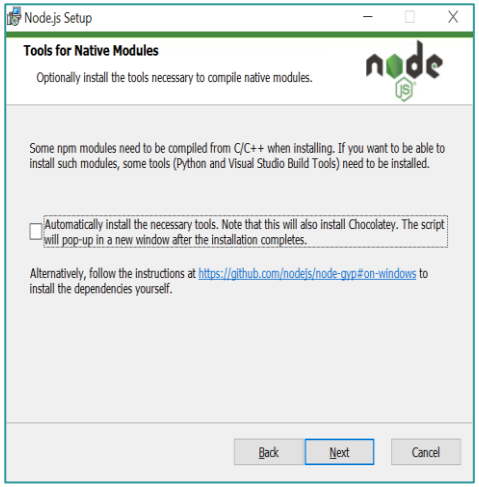


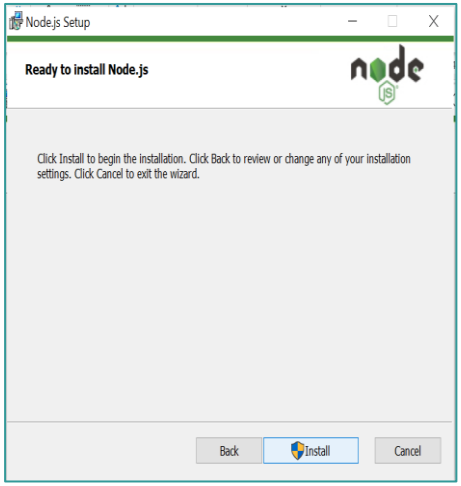




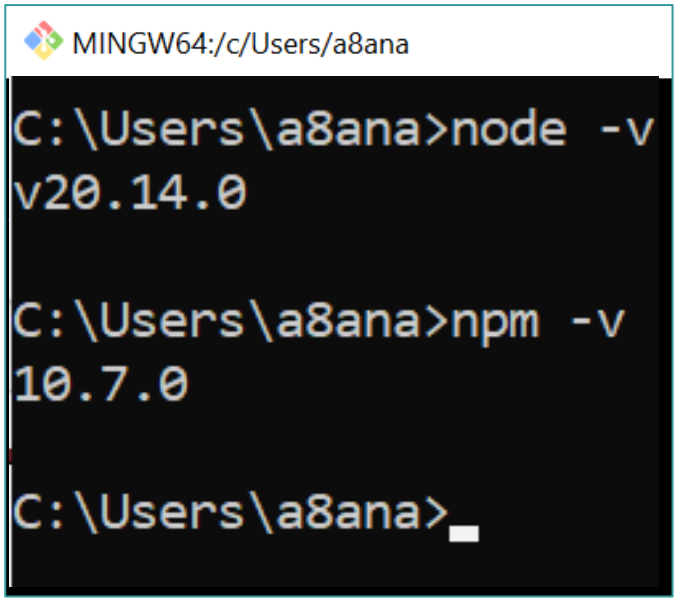








Επιβεβαίωση



- Επιβεβαιώνουμε το install με:
- node –v
- Και npm -v

- Περιλαμβάνεται στην εγκατάσταση το npm (node package manager), ένα εργαλείο που μας επιτρέπει να εγκαθιστούμε βιβλιοθήκες (packages) ώστε να χρησιμοποιούμε τη λειτουργικότητα που έχει αναπτυχθεί από τρίτους

VSCode και Node

1. Δημιουργούμε ένα .js αρχείο στο Visual Studio Code
2. Ανοίγουμε ένα terminal με View / Terminal (PowerShell ή Git Bash ή Command Prompt)
3. Κάνουμε cd στον φάκελο που είναι το .js αρχείο
4. Κάνουμε node helloworld.js (για να εκτελεστεί το helloworld.js)

Hello World στον Browser

1-js-intro > browser > helloworld.html > html > body

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-
6      <title>Hello World with JavaScript</title>
7  </head>
8  <body>
9
10     <script>
11         console.log("Hello World!")
12     </script>
13 </body>
14 </html>
```

- Ο κώδικας είμαι μέσα στο **<script> tag**
- Θα ήταν καλύτερα ωστόσο, να διαχωρίσουμε την HTML που είναι presentational με την JavaScript που είναι λογική της εφαρμογής (βλ. επόμενη διαφάνεια)

Hello World (2)

1-js-intro > browser > helloworld.html > ...

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-
6       <title>Hello World with JavaScript</title>
7   </head>
8   <body>
9
10      <script src="./hello.js"></script>
11  </body>
12  </html>
```

- Αντί για κώδικα στο <script> έχουμε στην ιδιότητα src, το αρχείο .js

Hello World (3)

hello.js

1-js-intro > browser > hello.js

```
1   console.log("Hello World!")
```

- Το hello.js έχει το console.log, το οποίο εκτελείται όπως πριν, αλλά τώρα έχουμε διαχωρίσει το *view* από τον κώδικα που διαχειρίζεται τη λογική της εφαρμογής (separation of concerns)

Hello World / Web Browser (1)

helloworld.html

1-js-intro > browser > helloworld.html > ...

```
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device
6       <title>Hello World with JavaScript</title>
7   </head>
8   <body>
9       <div>
10          <p id="helloText">
11
12          </p>
13      </div>
14
15      <script src="./hello-browser.js"></script>
16  </body>
17  </html>
```

hello-browser.js

1-js-intro > browser > hello-browser.js > ...

```
1   // Map HTML UI Element to DOM node
2   const pDOM = document.getElementById('helloText')
3   pDOM.innerHTML = "<strong>Hello World!</strong>"
4
5   // console for logging
6   console.log('Hello World!')
7
8
9
10
11
12
13
14
15
```

- Εισάγουμε το js αρχείο με το script tag, το οποίο και εκτελείται ακριβώς στο σημείο που βρίσκεται.
- Το <script> τυπικά είναι η τελευταία εντολή πριν το </body> ώστε να έχουν φορτωθεί στη μνήμη τα UI elements, ενώ επίσης δεν καθυστερεί το parsing της HTML

DOM (1)

1-js-intro > browser > JS hello-browser.js > ...

```
1  // Map HTML UI Element to DOM node
2  const pDOM = document.getElementById('helloText')
3  pDOM.innerHTML = "<strong>Hello World!</strong>"
4
5  // console for logging
6  console.log('Hello World!')
```

- Η JavaScript όπως θα δούμε, όσο αφορά το I/O επενεργεί επί των UI elements που έχουν φορτωθεί στη μνήμη. Θα πρέπει δηλαδή πρώτα να έχει φορτωθεί στη μνήμη αυτό που ονομάζουμε **DOM (Document Object Model)**, δηλαδή μία αναπαράσταση την HTML σελίδας σε δενδρική μορφή και μετά η JavaScript να έχει προγραμματιστική πρόσβαση στα UI Elements του DOM

DOM (2)

1-js-intro > browser > JS hello-browser.js > ...

```
1  // Map HTML UI Element to DOM node
2  const pDOM = document.getElementById('helloText')
3  pDOM.innerHTML = "<strong>Hello World!</strong>"
4
5  // console for logging
6  console.log('Hello World!')
```

- Για να πάρουμε ένα **reference σε ένα DOM node** όπως το <p id="helloText"> μπορούμε με **document.getElementById('helloText')** ή και με **document.querySelector('#helloText')** όπου document είναι το root node του DOM, άρα ψάχνει μέσα σε όλο το document να βρει το στοιχείο με συγκεκριμένο id

DOM (3) - Output

1-js-intro > browser > JS hello-browser.js > ...

```
1  // Map HTML UI Element to DOM node
2  const pDOM = document.getElementById('helloText')
3  pDOM.innerHTML = "<strong>Hello World!</strong>"
4
5  // console for logging
6  console.log('Hello World!')
```

- Όλα τα στοιχεία του DOM που αναπαριστούν UI elements με περιεχόμενο, μας παρέχουν την ιδιότητα **innerHTML** για να κάνουμε get/set το περιεχόμενο του στοιχείου προγραμματιστικά. Η innerHTML κάνει parse και HTML (άρα 'αναγνωρίζει' το)
- Οι κόμβοι του δένδρου (DOM nodes), στην JavaScript αναπαρίστανται ως objects, άρα το pDOM είναι ένα JS object που αναπαριστά το <p> element στην HTML)

DOM (4) – Input / Output

helloworld.html

1-js-intro > browser > helloworld.html > ...

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Hello World with JavaScript</title>
7  </head>
8  <body>
9      <div>
10         <label for="inputText">Hello label</label>
11         <input type="text" id="inputText" placeholder="Insert Hello Text to Echo">
12         <input type="button" id="submit" value="Submit" onclick="printText()">
13
14         <p id="helloText">
15
16         </p>
17     </div>
18
19     <script src="./hello-browser.js"></script>
20 </body>
21 </html>
```

- Για να λάβουμε input θα πρέπει να έχουμε ένα στοιχείο όπως το textbox (**<input type="text">**). Επίσης, θα πρέπει να υπάρχει ένα **event που να σηματοδοτεί ότι έχει δοθεί το κείμενο στο textbox** και να γίνει κάποια ενέργεια μέσω μίας συνάρτησης που κάνει handle το event (event handler). Το event αυτό είναι το click event που ενεργοποιεί ο χρήστης με το click στο button

DOM (5) – Input / Output

1-js-intro > browser > JS hello-browser.js > ...

```
1  function printText() {
2      const text = document.getElementById('inputText').value
3      document.getElementById('helloText').innerHTML = text
4  }
```

- Στο onclick του button καλείται η printText() που διαβάζει το *value* από το textbox με id="inputText" και γράφει στο innerHTML του <p> με id="helloText"
- Γράφει αυτό που έχει διαβάσει (echo)

DOM (6) – Input / Output


```
8  <body>
9      <div>
10         <label for="inputText">Hello label</label>
11         <input type="text" id="inputText" placeholder="Insert Hello Text to Echo"
12                oninput="printText()">
13
14         <p id="helloText">
15
16         </p>
17     </div>
18
19     <script src="./hello-browser.js"></script>
20 </body>
21 </html>
```

- Εδώ το event είναι διαφορετικό, είναι το **oninput** στο **textbox**, που ενεργοποιείται κάθε φορά που αλλάζει το input στο textbox

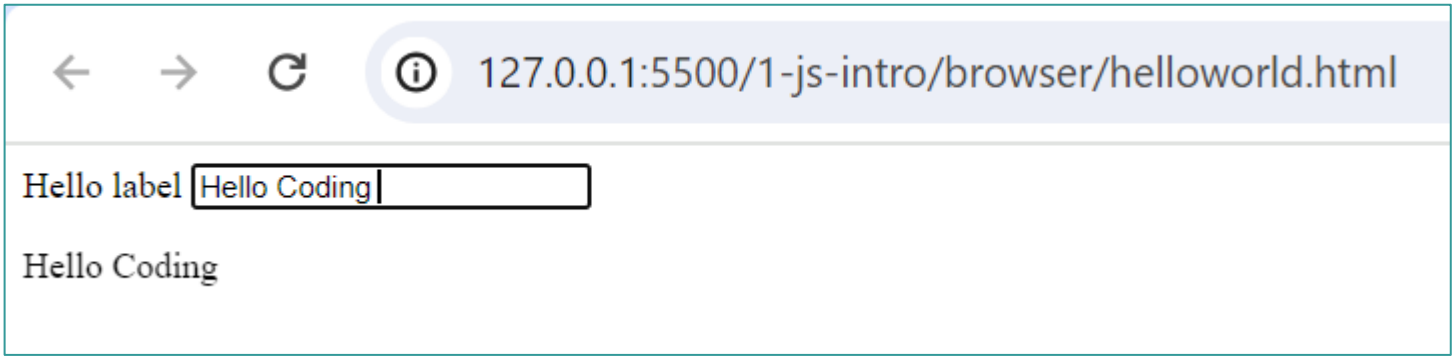
DOM (7) – Input / Output

1-js-intro > browser > JS hello-browser.js > ...

```
1  function printText() {
2      const text = document.getElementById('inputText').value
3      document.getElementById('helloText').innerHTML = text
4  }
```

- Κάθε φορά που θα αλλάζει το input από τον χρήστη, άμεσα θα αλλάζει και το output (echo effect)

DOM (8) – Input / Output



- Καθώς γράφουμε ταυτόχρονα αλλάζει το κείμενο στο <p>