



Προγραμματισμός με JavaScript

Αθανάσιος Ανδρούτσος



JS Data Type System

JavaScript

- Σύμφωνα με το τελευταίο πρότυπο ECMAScript υπάρχουν επτά (7) πρωταρχικοί τύποι στην JavaScript: **number**, **boolean**, **string**, **undefined**, **bigint**(ES2020), **symbol**, **null**
- Οτιδήποτε δεν είναι object στην JavaScript είναι primitive
- Οι πρωταρχικοί τύποι στην JavaScript είναι immutable



Primitives

Type	<code>typeof</code> 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



Number

- Πρόκειται για Floating point numbers που βασίζονται στο πρότυπο IEEE 754 (1 bit πρόσημο, 52 bits mantissa –δεκαδικός μεταξύ 0 και 1- και 11 bits εκθέτης –μεταξύ -1022 και 1023- όπως οι double της Java)

$$\text{Number} = (- 1)^{\text{sign}} \cdot (1 + \text{mantissa}) \cdot 2^{\text{exponent}}$$

- Αρχικοποιούμε numbers είτε **με τον constructor function Number()** και **new** (οπότε ο τύπος είναι object), ή κυρίως με number literals
- Με το **function Number()** **χωρίς new** μετατρέπουμε (coerce) strings σε number. Αν το string δεν είναι number επιστρέφεται NaN



Number Properties & Methods (1)

JavaScript

```
1 console.log("Min-Max Val: " + Number.MIN_VALUE + " - " + Number.MAX_VALUE)
2 console.log("Min - Max safe int: " + Number.MIN_SAFE_INTEGER + " - " + Number.MAX_SAFE_INTEGER)
3
4 console.log("Is ABS a number? : " + Number("ABC"))
5
6 console.log("Positive div by zero: " + (12/0))
7 console.log("Negative div by zero: " + (-12/0))
8
9 let num = Number("12") // Coercion of string to number
10 if (Number.isInteger(num)) {
11     console.log("Number.isInteger(12): " + "true")
12 } else {
13     console.log("Not an integer")
14 }
15
16 let aNum = 4 / "a"
17 if (Number.isNaN(aNum)) {
18     console.log("Number.isNaN(4/'a'): " + aNum)
19 } else {
20     console.log(aNum)
21 }
22
23 console.log("toFixed(2): " + (12.12345).toFixed(2))
```



Number Properties & Methods (2)

JavaScript

```
Min-Max Val: 5e-324 - 1.7976931348623157e+308
```

```
Min - Max safe int: -9007199254740991 - 9007199254740991
```

```
Is ABS a number? : NaN
```

```
Positive div by zero: Infinity
```

```
Negative div by zero: -Infinity
```

```
Number.isInteger(12): true
```

```
Number.isNaN(4/'a'): NaN
```

```
.toFixed(2): 12.12
```

- Αποτέλεσμα του προηγούμενου παραδείγματος με τη χρήση static properties, static μεθόδων και instance μεθόδων του τύπου Number



Boolean (1)

- Αναπαριστά μία τιμή αλήθειας: true ή false
- Μπορούμε να πάρουμε άμεσα boolean με
 - Boolean literal. π.χ. `let isSenior = true`
 - Boolean constructor function χωρίς `new` ή `double NOT`. π.χ. `let b = Boolean("Hello") // b === true`
ή `let b = !! "Hello" // b === true`
 - Boolean constructor function με `new`. π.χ. `let b = new Boolean(false)`



Boolean (2)

- Οι μετατροπές άλλων τύπων δεδομένων σε boolean με την function Boolean() μετατρέπει κάποιες τιμές σε false (falsy values). Σε boolean context οι μετατροπές αυτές γίνονται αυτόματα
 - Booleans coerce as-is
 - Undefined, null -> false
 - 0 -0, 0n, NaN -> false
 - "", "", `` -> false



Boolean (2)

- Όλες οι άλλες μετατροπές δίνουν truthy values
- Για παράδειγμα όλα τα objects μετατρέπονται σε true, ακόμα και τα boolean objects που είναι false, π.χ. `let b = !(new Boolean(false))` ή `!!{}`
- Booleans επίσης δίνουν τα expressions με τελεστές σύγκρισης (`>`, `>=`, `<`, `<=`, `==`, `===`, `!=`, `!==`)
- Boolean contexts είναι οι δομές ελέγχου `if`, `while`, `for`, `switch`



Boolean (3)

JavaScript

```
1 console.log("Boolean(1): " + Boolean(1))
2 console.log("Boolean(0) : " + Boolean(0))
3 console.log('Boolean("") : ' + Boolean(""))
4 console.log('Boolean("Hello") : ' + Boolean("Hello"))
5 console.log("Boolean(null) : " + Boolean(null))
6 console.log("Boolean(undefined) : " + Boolean(undefined))
7 console.log("Boolean({}) : " + Boolean({}))
8 console.log("Boolean(new Boolean(false)) : " + Boolean(new Boolean(false)))
9
10 console.log((0) ? "1 is True" : (0) ? "0 is true" : "0 is False")
11
12 while (1) { break }
13 do { console.log("Hello Coding Factory") } while (0)
14 for (i = 1; 1; i++) { console.log("In for"); break }
```

- Μερικά παραδείγματα χρήσης boolean



Boolean (3)

```
Boolean(1): true
Boolean(0) : false
Boolean("") : false
Boolean("Hello") : true
Boolean(null) : false
Boolean(undefined) : false
Boolean({}) : true
Boolean(new Boolean(false)) : true
0 is False
Hello Coding Factory
In for
```

- Falsy & Truthy values
- Αποτέλεσμα του προηγούμενου παραδείγματος



Strings (1)

- Τα strings στην JavaScript, είναι zero-based indexed arrays από 16-bit Unicode codepoints
- Το string θεωρείται πρωταρχικός τύπος στην JS (δεν είναι object) και είναι immutable
- Στην JS δεν υπάρχει τύπος δεδομένων char. Μοναδικούς 16-bit χαρακτήρες, αναπαριστούμε με strings μήκους 1
- Τα empty strings έχουν length 0



Strings (2)

- Επομένως κάθε string είναι μία ακολουθία 16-bit Unicode characters (UTF-16)
- Τα Unicode characters του basic multilingual plane έχουν codepoints που χωρούν σε 16-bits και μπορούν να αναπαρασταθούν στην JavaScript με ένα string μήκους 1



Strings (3)

- Τα Unicode characters που δεν χωρούν σε 16-bits μπορούν να αναπαρασταθούν ως surrogate pairs με δύο 16-bit values, δηλαδή ένα string μήκους 2
- Οι περισσότερες string methods (π.χ. απλή for) λειτουργούν σε 16-bit values και όχι σε surrogate pairs
- Στην ES6 και μετά, τα strings είναι **Iterable** με την **for/of** και τον **spread operator (...)** ενώ γίνονται και parse οι κανονικοί χαρακτήρες, όχι τα 16-bit values



For .. of

```
> let s = "AUEB"  
undefined  
> for (const letter of s) {  
    console.log(letter)  
}  
A  
U  
E  
B
```

- Με for .. of μπορούμε να κάνουμε traverse strings ή πίνακες και γενικά ότι είναι iterable



String Literals

- Όπως έχουμε δει τα sting literals δίνονται είτε μέσα σε **double quotes** ή **single quotes**
- Επίσης, μπορούμε να χρησιμοποιήσουμε **template literals (ES6)** μέσα σε **backticks**, οπότε μπορούμε και να ενθέσουμε μεταβλητές (interpolation) με **`${}`**
- Όταν ένα string καταλαμβάνει πολλές γραμμές, μπορούμε να χρησιμοποιούμε `\` στο τέλος της γραμμής, να συνενώνουμε με `+` ή ακόμα καλύτερα να χρησιμοποιούμε **backticks**



Escape sequences

JavaScript

- Ισχύουν τα escape sequences

Sequence	Character represented
<code>\0</code>	The NUL character (<code>\u0000</code>)
<code>\b</code>	Backspace (<code>\u0008</code>)
<code>\t</code>	Horizontal tab (<code>\u0009</code>)
<code>\n</code>	Newline (<code>\u000A</code>)
<code>\v</code>	Vertical tab (<code>\u000B</code>)
<code>\f</code>	Form feed (<code>\u000C</code>)

Sequence	Character represented
<code>\r</code>	Carriage return (<code>\u000D</code>)
<code>\"</code>	Double quote (<code>\u0022</code>)
<code>\'</code>	Apostrophe or single quote (<code>\u0027</code>)
<code>\\</code>	Backslash (<code>\u005C</code>)



Δημιουργία String

JavaScript

```
> let s1 = String(12) // String constructor called as a function performs coercion
< undefined

> let s2 = new String("Hello") // String constructor called with new returns a String object (of type String wrapper)
< undefined

> let s3 = "Hello Coding Factory" // String init with string literal - double quotes
< undefined

> let s4 = 'Hello All' // String init with string literal - single quotes
< undefined

> let s5 = `Hello
Coding
Factory` // Template Strings may include newlines
```

- Με String constructor function, με String constructor function και new, και συνήθως με *string literal*
- Με String constructor function και new ο επιστρεφόμενος τύπος είναι object
- Προτιμότερος τρόπος είναι η δημιουργία με string literal



String Parsing functions

JavaScript

- Επιστρέφουν μέρος του String (tokens)
- Λέγοντας *parse* γενικά εννοούμε τον διαχωρισμό ενός input string σε επιμέρους tokens
- Η `substring()` και `slice()` δίνουν `endIndex-1` ενώ η 2^η παράμετρος της `substr()` είναι το `length` (η **`substr` είναι πλέον deprecated**)
- Η `split()` διαχωρίζει με βάση ένα delimiter και επιστρέφει πίνακα. Μπορεί να λάβει RegEx. Π.χ. `s.split(/[^\w]/)` non-word characters (letters, digits, underscore)

```
> let s = 'Coding Factory'
undefined
> s.substr(1, 5)
'oding'
> s.substring(1, 5)
'odin'
> s.slice(1, 5)
'odin'
> s.slice(-7)
'Factory'
> s.split(" ")
[ 'Coding', 'Factory' ]
```



String functions - search

JavaScript

- Search functions
 - `charAt()`
 - `codePointAt()`
 - `indexOf()`
 - Η θέση της 1^{ης} εμφάνισης
 - `lastIndexOf()`
 - Η θέση της τελευταίας εμφάνισης

```
> let s = "Coding Factory"
undefined
> s.charAt(1)
'o'
> s.codePointAt(0)
67
> s.indexOf('o')
1
> s.lastIndexOf('o')
11
> s.indexOf('o', 2)
11
```



Replace functions

JavaScript

- Αντικαθιστούν μέρη ή όλο το String
- Αφού τα strings είναι immutable οι συναρτήσεις που επιστρέφουν νέο String, δεν επηρεάζουν το αρχικό
- Μπορούμε να **συγκρίνουμε case insensitive strings** με
`s1.toUpperCase() === s2.toUpperCase`

```
> s.replace("Coding", "Code")  
'Code Factory'  
> s.toLowerCase()  
'coding factory'  
> s.toUpperCase()  
'CODING FACTORY'
```



Διάφορες functions

JavaScript

- trim() – αποκόπτει τα κενά στην αρχή και το τέλος του string
- ES2019
 - trimStart()
 - trimEnd()
- Συνένωση
 - concat()
 - Τελεστής +
- repeat

```
type 'help' for more information
> let s = "Coding Factory  "
undefined
> s.trim()
'Coding Factory'
> "Hello " + s
'Hello Coding Factory  '
> "Hello " + s.trim()
'Hello Coding Factory'
> "Hello ".concat(s.trim())
'Hello Coding Factory'
> "Hello ".repeat(5)
'Hello Hello Hello Hello Hello '
```



Text processing – pattern matching

JavaScript

```
1  const text = "05/01/2023"
2  const datePattern = /\d{2}/g
3
4  const matches = text.match(datePattern)
5
6  console.log(matches)
```

node regex.js

```
PS C:\Users\A8ana\OneDrive\
[ '05', '01', '20', '23' ]
```

- Ο μηχανισμός pattern matching είναι μέρος των Κανονικών Εκφράσεων (Regular Expressions)
- Στην JavaScript κανονικές εκφράσεις γράφουμε μέσα σε / / και μπορούμε να ελέγχουμε με .match(pattern)
- Με /g (global) ψάχνει για όλα τα occurrences



Groups

```
1 const text = "05/01/2023 04/12/2024"
2 const datePattern = /(\d{2})\/(\d{2})\/(\d{4})/g
3
4 let match
5 while ((match = datePattern.exec(text)) !== null) {
6     [fullDate, day, month, year] = match
7     console.log(`full: ${fullDate}, day: ${day}, month: ${month}, year: ${year}`)
8 }
```

- Όταν έχουμε groups μπορούμε να πάρουμε πίνακα για το κάθε occurrence με .exec
- Κάνουμε dispatch τον επιστρεφόμενο πίνακα στις μεταβλητές fullDate (match[0]), day (match[1]), month (match[2]), year (match[3]) για κάθε match



Global Object (1)

- Το Global Object είναι ένα ειδικό object που έχει global scope και παρέχει όλο το ECMAScript/JavaScript API (JavaScript library), δηλαδή όλα τα variables, constants και functions που είναι διαθέσιμα προς χρήση
- Τεχνικά είναι ένα απλό object με properties & methods
- Στον browser το Global Object φορτώνεται μόλις ανοίξει ένα παράθυρο (tab) και ονομάζεται *window*. Σε περιβάλλον node επίσης δημιουργείται από τον Interpreter και ονομάζεται *global*
- Η ES2020 όρισε το ***globalThis*** ως τον standard τρόπο αναφοράς στο global object σε οποιοδήποτε host environment (browser, node ή αλλού)



Global Object (2)

- Το global object περιλαμβάνει:
 - Global constants όπως **undefined**, **NaN**, **Infinity** (το *null* είναι keyword της γλώσσας και όχι global constant)
 - Global functions όπως **isNaN()**, **parseInt()**, **parseFloat()**
 - Constructor functions όπως **Date()**, **RegExp()**, **String()**, **Object()**, **Array()**
 - Global objects όπως **Math**, **JSON**



Global Object (3)

- Στον browser, οι δηλώσεις **var** έξω από συναρτήσεις (global vars) δημιουργούνται ως members του global object (όχι στο περιβάλλον Node.js)
- Άλλα objects που ανήκουν στο global scope είτε δημιουργούνται από τα **user scripts** ή παρέχονται από το **host environment**
- Για παράδειγμα, τα function declarations στο top level ενός αρχείου .js δημιουργούνται ως methods του global object



Σύνθετες δομές – Πίνακες

JavaScript

- Οι πίνακες στην JS είναι untyped, zero-based διατεταγμένες ακολουθίες στοιχείων, οποιουδήποτε τύπου, primitive ή reference type
- **Untyped**, σημαίνει ότι τα στοιχεία του πίνακα μπορεί να μην είναι του ίδιου τύπου, άρα τα arrays στην JS μπορεί να περιέχουν ένα **mix από διάφορα data types**
- Τα arrays στην JS είναι special objects και τα στοιχεία τους μπορούν να προσπελαστούν με μη αρνητικά indexes. Τα indexes των πινάκων είναι 32-bit από 0 – $2^{32}-2$.



Dynamic Arrays (1)

- Στην JS **τα arrays είναι dynamic**. Μεγαλώνουν (**grow**) και μικραίνουν (**shrink**) όταν χρειάζεται (resizable). Επομένως δεν χρειάζεται να ορίζουμε size στα arrays
- Κάθε array έχει ένα length property.
- Τα Arrays ωστόσο μπορεί να είναι *sparse*, δηλαδή τα array elements μπορεί να μην είναι συνεχόμενα και να υπάρχουν κενά. Οπότε τότε το length του array μπορεί να είναι μεγαλύτερο από το πλήθος των στοιχείων



Dynamic Arrays (2)

- Στα JS Arrays δεν υπάρχει `IndexOutOfRangeException` όπως στη Java
- Αν προσπελάσουμε μία θέση μεγαλύτερη από το τελευταίο στοιχείο ή μία θέση μεταξύ των στοιχείων σε `sparse arrays` παίρνουμε `undefined`
- Επίσης αν πάμε να αποθηκεύσουμε ένα στοιχείο σε θέση μεγαλύτερη από το τελευταίο στοιχείο, το `array` θα γίνει `expand` και το στοιχείο θα αποθηκευτεί



Arrays as Objects (1)

- Τα **arrays** υλοποιούνται ως **special objects** στην **JS**. Τα **indexes** είναι στην πραγματικότητα **property names** που τυγχάνει να είναι **integers**
- Ωστόσο δεν μπορούμε να προσπελάσουμε ένα **array object** με **dot notation**, **arr.0** αλλά με **bracket notation**, **arr[0]** . Το **0** αυτόματα μετατρέπεται σε **"0"**



Arrays as Objects (2)

JavaScript

```
1 const arr = [1, 2, 3];  
2  
3 const arrAsObject = {  
4   0: 1,  
5   1: 2,  
6   2: 3,  
7   length: 3,  
8   // Other array-specific properties and methods  
9   // ...  
10  };
```

- Η πραγματική αναπαράσταση στη μνήμη ενός array είναι ως object



Arrays as Objects (3)

JavaScript

- Στο bracket notation μπορούμε να αναφερθούμε εκτός από `arr[1]` για παράδειγμα και ως `arr['1']` ωστόσο αυτό δεν χρειάζεται
- Η JS αυτόματα μετατρέπει το `arr[1]` σε `arr['1']` και για αυτό το `arr[1] !== arr[01]`. Το `arr[01] === arr['01']` και δεν αποτελεί έγκυρη θέση του πίνακα



Arrays as Objects (4)

JavaScript

- Τα arrays έχουν properties και methods και **όλα τα arrays κληρονομούν έμμεσα** (extends) από το **Array.prototype** που είναι το όνομα της root κλάσης στην ιεραρχία των Arrays
- Η Array.prototype ορίζει πολλές μεθόδους διαχείρισης arrays τις οποίες κληρονομούν όλα τα arrays



Δημιουργία Arrays (1)

JavaScript

- Με **Arrays literal**
 - `let arr = [1, 2, 3]`
 - `let arr = []` `// κενό array`
- Με **Array() constructor function**
 - `let arr = Array(2)` `// [undefined, undefined]`
 - `let arr = new Array(2)` `// [undefined, undefined]`
το `new` είναι optional
 - `let arr = Array(1, 2, 3)` `// [1, 2, 3]`
 - `let arr = new Array(1, 2, 3)` `// [1, 2, 3]`
το `new` είναι optional



Δημιουργία Arrays (2)

JavaScript

- Με ... spread operator. Τα 3 dots, κάνουν spread τον array
 - Αν έχουμε `arr = [1, 2, 3]`, τότε το **`arr2 = [...arr]`** είναι ένα αντίγραφο του `arr`. Είναι shallow copy (όλα τα JS functions δημιουργούν shallow copies) Το spread operator δουλεύει σε Iterable objects
- Με τις `Array.of()` και `Array.from()` factory methods



Δημιουργία Arrays (2)

JavaScript

```
> let arr = [] // Empty Array
< undefined

> let primes = [2, 3, 5, 7, 11, 13] // Array with 6 elements
< undefined

> let misc = [5, 1.5, false, "Hello", ] // Array with 4 different elements and trailing comma
< undefined

> let grid = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
< undefined

> let grid = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] // 3x3 2D Array
< undefined

> let objArr = [{id: 1, name: "Honey", price: 12.8}, {id: 2, name: "Milk", price: 1.5}] // array of objects
< undefined

> let count = [1,,3] // Sparse array, no element at position 1
< undefined

> let undefs = [,] // undefs.length === 2. optional trailing comma
< undefined
```

- Δημιουργία με array literal. Αν θέσουμε ***arr.length = 0*** διαγράφουμε όλα τα elements του array και γενικά αν θέσουμε ***arr.length = n***, όπου $n < \text{τρέχον arr.length}$ διαγράφουμε τα στοιχεία από n και πάνω



Δημιουργία Arrays (3)

JavaScript

```
> let initial = [1, 2, 3]
< undefined

> let postInitial = [0, ...initial, 4] // // [0, 1, 2, 3, 4]
< undefined

> let chars = [..."Coding Factory"]
< undefined

> chars
< ► (14) ['C', 'o', 'd', 'i', 'n', 'g', ' ', 'F', 'a', 'c', 't', 'o', 'r', 'y']
>
```

- Ο ... spread operator δουλεύει μόνο σε Iterable objects. Τα arrays είναι Iterable



Δημιουργία Arrays (4)

JavaScript

```
> let arr = Array(5) // declares an array of 5 undefined elements
```

```
< undefined
```

```
> let arr = new Array(5) // same as above. Array constructor function can be used with or without new
```

```
< undefined
```

```
> let arr = Array(1, 2, 3, 4, 5) // Array of length 5 with specific elements
```

```
< undefined
```

```
> let arr = new Array(1, 2, 3, 4, 5) // the same as above (with new and constructor function)
```

```
< undefined
```

```
> let arr = new Array(3)
```

```
> for (let i = 0; i < arr.length; i++) {  
  arr[i] = new Array(2)  
  for (let j = 0; j < arr[i].length; j++) {  
    arr[i][j] = `[${i} ${j}]`  
  }  
}
```

```
// 2D array on arr - each element of arr is an array of 2 elements
```

- Multi-dimensional array

- Δημιουργία με Array constructor function



Array.of & Array.from static Array methods

JavaScript

```
> let arr = Array.of(5)
```

```
< undefined
```

```
const arr1 = Array.of(1, 2, 3);  
console.log(arr1); // [1, 2, 3]
```

```
const arr2 = Array.of(5);  
console.log(arr2); // [5]
```

```
> let arr2 = Array.from(arr)  
// .from gets an iterable and transforms (and copies) it to array  
// works like spread operator. Both spread operator and .from static  
// method work also on array-like objects
```

```
< undefined
```

- Ο Array constructor δεν μπορεί να δημιουργήσει arrays με 1 στοιχείο. Η Array.of() (ES6) μπορεί να διορθώσει αυτό το θέμα
- Η Array.from() δημιουργεί array-copies από iterable objects (strings, sets, maps) ή array-like objects, όπως NodeList). Τα array-like objects είναι objects (όχι arrays) που έχουν ένα length property και integer indexes
- Η Array.from() μπορεί να πάρει και ως 2^η παράμετρο μία callback function που να κάνει map κάθε τιμή σε μία άλλη (όπως και η .map)
`const arr = Array.from([1, 2, 3], x => x * 2); // [2, 4, 6]`



Πράξεις σε πίνακες

JavaScript

- Διάσχιση των στοιχείων του πίνακα
- Εισαγωγή, Διαγραφή, Αλλαγή στοιχείων ή subarrays (splice)
- Αναζήτηση ενός στοιχείου, Ταξινόμηση ενός πίνακα (sort)
- Stack & Queue λειτουργίες (push, pop, unshift, shift)



Traversing arrays (1)

JavaScript

- Μπορούμε να διασχίσουμε ένα πίνακα με τρεις τρόπους
 - Κλασσική for
 - for / of (ES6)
 - Την μέθοδο `forEach()`



Traversing arrays (2)

JavaScript

```
> let arr = [1, 2, 3, 4, 5, 6,,8, 9, 10] // Sparse Array, length is 10, arr[6] is undefined
```

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

1, 2, 3, 4, 5, 6, undefined, 8, 9, 10,
< undefined
```

```
> let out = ""
  for (let item of arr) {
    out += item + ", "
  }
  console.log(out)

1, 2, 3, 4, 5, 6, undefined, 8, 9, 10,
< undefined
```

- Με κλασσική **for** ή με **for / of** μπορούμε και κάνουμε iterate τον Array.
- Εμφανίζονται και τα **undefined** στοιχεία



forEach method (1)

JavaScript

```
> arr.forEach(el => console.log(el))  
1  
2  
3  
4  
5  
6  
8  
9  
10  
← undefined
```

- Η `forEach()` λαμβάνει υπόψη τα `sparse arrays` και δεν εκτυπώνει τα `undefined elements`

- Στην ES6 έχουμε και την **`forEach()`** instance method. Επειδή ο κώδικας για τη διαχείριση arrays είναι verbose και boilerplate θα θέλαμε ιδανικά να μπορούμε να μετατρέψουμε τα iterations των `for` σε functional programming
- Η **`forEach()`** παίρνει ως παράμετρο ένα **callback function** (εδώ σε μορφή `arrow function`) που επεξεργάζεται (καταναλώνει ή συνήθως εκτυπώνει) κάθε στοιχείο του array



forEach method (2)

JavaScript

```
> let out = ""  
   arr.forEach(elem => out += elem + ", ")  
   console.log(out)  
  
   1, 2, 3, 4, 5, 6, 8, 9, 10,  
  
◀ undefined
```

- Στο σώμα της `forEach` μπορούμε να κάνουμε και υπολογισμούς, μετατροπές, κλπ.
- Εδώ υπολογίζουμε το `out`



concat

JavaScript

```
> let arr1 = [1, 2, 3]
< undefined
> arr1.concat(4, [5, 6])
< ► (6) [1, 2, 3, 4, 5, 6]
```

- Η concat δεν κάνει modify το array στο οποίο καλείται. Δημιουργεί ένα καινούργιο copy
- Αν θέλουμε να κάνουμε modify τον ίδιο πίνακα χρησιμοποιούμε την push ή την splice



Delete / add

JavaScript

```
let grades = [8, 6, 9, 2]
delete grades[0]
console.log(grades)
```

▼ (4) [empty, 6, 9, 2] ⓘ

- 1: 6
- 2: 9
- 3: 2
- length: 4

```
> grades
< ▶ (4) [empty, 6, 9, 2]
```

```
> grades.push(10)
< 5
```

```
> grades
< ▶ (5) [empty, 6, 9, 2, 10]
```

- Η **delete** δημιουργεί parse arrays, δεν μειώνεται το length. Η **push()** προσθέτει στο τέλος
- Ο γενικός τρόπος για να κάνουμε insert, delete και update στοιχεία σε ένα πίνακα είναι με τη μέθοδο **splice()**



Splice (1)

- Η `splice()` είναι μία γενική μέθοδος που επιτρέπει **insert, remove, replace**
- Τεχνικά κάνει remove 'slices' (subarrays) από το αρχικό array και (προαιρετικά) τα αντικαθιστά
- Η 1^η παράμετρος είναι το *startIndex* του slice, η 2^η παράμετρος είναι το *πλήθος των στοιχείων* που κάνουμε slice out (αν παραλείπεται υπονοείται ότι όλα τα στοιχεία γίνονται delete) Μετά τη 2^η παράμετρο μπορεί να ακολουθεί μία λίστα προαιρετικών τιμών insert



Splice (2) - Delete

JavaScript

```
> const arr = [1, 2, 3, 4, 5, 6, 7, 8]
< undefined
> const arr2 = arr.splice(0, 4)
< undefined
> arr2
< ▶ (4) [1, 2, 3, 4]
> arr
< ▶ (4) [5, 6, 7, 8]
```

- Delete από τη θέση με index 0 τέσσερα στοιχεία. Η splice επιστρέφει πίνακα με τα διαγραφμένα στοιχεία
- Επίσης κάνει modify τον αρχικό πίνακα

```
< ▶ (4) [5, 6, 7, 8]
> arr.splice(2, 1) // [7], arr is now [5, 6, 8]
< ▶ [7]
> arr
< ▶ (3) [5, 6, 8]
```

Delete από τη θέση με index 2 ένα στοιχείο



Splice (3) – Insert / Update

JavaScript

```
> arr.splice(1, 0, 17) // 0 items deleted from position 1. At this position add 17.
                        // All other elements (6, 8) will shift right
< ▶ []
> arr
< ▶ (4) [5, 17, 6, 8]
```

- Delete από τη θέση με index 1 **μηδέν στοιχεία** και πρόσθεσε 1 στοιχείο (**ουσιαστικά πρόκειται για insert**)

```
< ▶ (4) [5, 17, 6, 8]
> arr.splice(1, 1, 12) // delete and insert at position index 1
< ▶ [17]
> arr
< ▶ (4) [5, 12, 6, 8]
```

Delete από τη θέση με index 1, ένα στοιχείο και πρόσθεσε 1 στοιχείο (**ουσιαστικά πρόκειται για update**)



Slice, fill

```
> let arr = [1, 2, 3, 4, 5]
```

```
< undefined
```

```
> let sliced = arr.slice(0, 2) // slices out [1, 2]: first element => start-index: 0,  
// endElement => end-index - 1 that is 2 - 1 = index 1
```

```
< undefined
```

```
> sliced
```

```
< ▶ (2) [1, 2]
```

- Εξάγει ένα slice Αν παραληφθεί το endIndex υπονοείται το τέλος του πίνακα

```
> let arr2 = Array(5)
```

```
< undefined
```

```
> arr2.fill(0) // fill with zeros
```

```
< ▶ (5) [0, 0, 0, 0, 0]
```

```
> arr2.fill(5, 1) // fill with 5, start-index is 1
```

```
< ▶ (5) [0, 5, 5, 5, 5]
```

```
> arr2.fill(8, 2, 3) // fill with 8, start-index 2, end-index 3-1 = 2
```

```
< ▶ (5) [0, 5, 8, 5, 5]
```

- Fills ένα slice με μία τιμή. Αν παραληφθεί το index υπονοείται όλος ο πίνακας
- Αν παραληφθεί το end-index υπονοείται το τέλος του πίνακα
- Το τελευταίο στοιχείο είναι endIndex - 1



copyWithin

JavaScript

```
> let arr = [1, 2, 3, 4, 5]
< undefined

> arr.copyWithin(0, 2) // [3, 4, 5, 4, 5] copies elements with startIndex 2 till the end at position 0
< ▶ (5) [3, 4, 5, 4, 5]

> arr.copyWithin(3, 0, 2) // [3, 4, 5, 3, 4] copies at position index 3 the elements 3, 4 (index 0 - 1)
< ▶ (5) [3, 4, 5, 3, 4]
```

- Η 1^η παράμετρος είναι το destination index. Οι επόμενες δύο παράμετροι είναι το slice. Αν παραλείπεται η τελευταία υπονοείται.
- Μπορεί η 2^η παράμετρος να έχει και αρνητικές τιμές, οπότε είναι το offset από το τέλος



Stack

JavaScript

```
> let arr = [1, 2, 3, 4]
< undefined

> // implementation of a stack with push & pop
< undefined

> arr.push(5, 6, 7)
< 7

> arr
< ▶ (7) [1, 2, 3, 4, 5, 6, 7]

> let x = arr.pop()
< undefined

> x
< 7

> arr
< ▶ (6) [1, 2, 3, 4, 5, 6]
```

- Η `push()` εισάγει ένα ή περισσότερα στοιχεία στο τέλος του πίνακα
- Η `pop()` εξάγει από το τέλος ένα στοιχείο



Queue

```
> let arr = [1, 2, 3, 4]
< undefined

> // implementation of a queue with push (insert at the end) and shift (extract from front)
< undefined

> arr.push(5)
< 5

> arr
< ► (5) [1, 2, 3, 4, 5]

> let x = arr.shift()
< undefined

> x
< 1

> arr
< ► (4) [2, 3, 4, 5]
```

- Η `push()` εισάγει ένα ή περισσότερα στοιχεία στο τέλος του πίνακα. Η `shift()` εξάγει από την αρχή του πίνακα ένα στοιχείο



Εισαγωγή στην αρχή

JavaScript

```
< ▶ (4) [2, 3, 4, 5]  
> arr.unshift(0)  
< 5  
> arr  
< ▶ (5) [0, 2, 3, 4, 5]  
>
```

- Εισαγωγή στην αρχή κάνουμε με unshift



Searching

JavaScript

```
> let arr = [1, 2, 3, 4, 1, 2]
< undefined

> let pos = arr.indexOf(2)
< undefined

> pos
< 1

> let pos2 = arr.lastIndexOf(2)
< undefined

> pos2
< 5

> let pos10 = arr.indexOf(10) // returns -1 when element not exists
< undefined

> pos10
< -1
```

```
> arr.includes(4)
< true
```

- `indexOf()`
- `lastIndexOf()`
- `Includes()`
- Οι 2 πρώτες μέθοδοι με συγκρίσεις αναζητούν το index ενός στοιχείου
- Η 3^η μέθοδος επιστρέφει `true/false` αν υπάρχει ή όχι η τιμή που αναζητούμε στον πίνακα



Παράδειγμα

JavaScript

```
1  /**
2   * Returns an array of indexes with the occurrences
3   * of a value in an array.
4   *
5   * @param {Array} arr - the source array.
6   * @param {Number} x - the element to compare.
7   * @returns an array with the indexes where the element is found.
8   */
9  function getIndexes(arr, x) {
10     const results = []
11
12     arr.forEach((element, index) => {
13         if (element === x) {
14             results.push(index)
15         }
16     })
17
18     return results
19 }
20
21 const arr = [1, 2, 3, 4, 5, 4, 3, 4]
22 console.log(getIndexes(arr, 4))
```

- Αναζητά σε ένα πίνακα `arr` ένα στοιχείο `x` και εισάγει σε ένα πίνακα `results` όλες τις εμφανίσεις (τα `indexes`) του `x`



Arrays Sort (1)

JavaScript

```
> let arr = ["Honey", "Milk", "Apples"]  
< undefined  
  
> arr.sort()  
< ▶ (3) ['Apples', 'Honey', 'Milk']  
> |
```

```
> let arr2 = [14, 123, 2, 6, 690, 60]  
< undefined  
  
> arr2.sort()  
< ▶ (6) [123, 14, 2, 6, 60, 690]  
> |
```

- Η `sort()` χωρίς παραμέτρους ταξινομεί με βάση το UTF-16 codepoints σε ascending order
- Και ενώ τα strings ταξινομούνται σωστά, οι υπόλοιποι τύποι δεδομένων μετατρέπονται σε strings και όπως φαίνεται οι αριθμοί δεν ταξινομούνται σωστά δεδομένης της μετατροπής σε UTF-16



Arrays Sort (2)

JavaScript

- Η `sort()` μπορεί να λάβει και μία παράμετρο που είναι ένα `callback function`, μία συνάρτηση ταξινόμησης που ορίζει το `sort order`
- Αν ονομάσουμε αυτή τη συνάρτηση `compare(a, b)` όπου `a, b` είναι δύο στοιχεία του πίνακα, τότε αν $a > b$ η συνάρτηση επιστρέφει `1`, αν $a < b$ επιστρέφει `-1` και αν $a == b$ επιστρέφει `0`
- Αυτή η λογική μπαίνει έμμεσα σε μία συνάρτηση ταξινόμησης (π.χ. `Bubble sort`) η οποία όταν είναι $a > b$ (δηλαδή $a - b > 0$) ταξινομεί `a` after `b`, όταν είναι $a < b$ (δηλαδή $a - b < 0$) ταξινομεί `a` before `b` και όταν είναι $a === b$ αφήνει το `original order`



Arrays Sort (3)

JavaScript

```
> let arr = [14, 123, 2, 6, 690, 60]
arr.sort((a, b) => a - b)
console.log(arr)

▶ (6) [2, 6, 14, 60, 123, 690]
< undefined
```

```
> let arr = [14, 123, 2, 6, 690, 60]
arr.sort(function(a, b) {
  return a - b
})
console.log(arr)

▶ (6) [2, 6, 14, 60, 123, 690]
< undefined
```

- Τα δύο snippets είναι όμοια, το μόνο που αλλάζει είναι ότι στην 1^η περίπτωση έχουμε arrow function ως callback στην sort ενώ στην 2^η περίπτωση ανώνυμη συνάρτηση
- Σε κάθε περίπτωση εκφράζουμε την λογική που της συνάρτησης compare που αναφέραμε (βλ. επόμενη διαφάνεια)



Arrays Sort (4)

JavaScript

```
1 let arr = [14, 123, 2, 6, 690, 60]
2
3 arr.sort(function(a, b) {
4     if (a > b) return 1
5     else if (a < b) return -1
6     else return 0
7 })
8
9 console.log(arr)
```

```
> let arr = [14, 123, 2, 6, 690, 60]
   arr.sort(function(a, b) {
       return a - b
   })
   console.log(arr)

▶ (6) [2, 6, 14, 60, 123, 690]

< undefined
```

- Τα δύο παραπάνω snippets είναι ίδια. Το `a-b` είναι σύντομος κώδικας. Αν κάνουμε `return a-b` τότε επιστρέφουμε `> 0` αν `a > b`, `< 0` αν `a < b` και `0` αν `a === b`.



Reverse

JavaScript

```
> let arr = [14, 123, 2, 6, 690, 60]
arr.sort((a, b) => b - a)
console.log(arr)

▶ (6) [690, 123, 60, 14, 6, 2]

< undefined
```

- Με $b - a$ παίρνουμε φθίνουσα (descending) κατάταξη

```
> let arr = [14, 123, 2, 6, 690, 60]
arr.sort((a, b) => a - b)
console.log(arr)

▶ (6) [2, 6, 14, 60, 123, 690]

< undefined

> arr.reverse()
< ▶ (6) [690, 123, 60, 14, 6, 2]
```

- Το ίδιο αποτέλεσμα μπορούμε να έχουμε αν κάνουμε ascending sort και μετά εφαρμόσουμε τη συνάρτηση `reverse()`



Sorting objects

JavaScript

```
> let products = [{name: "Milk", price: 10}, {name: "Honey", price: 16},  
                  {name: "Oranges", price: 4}, {name: "Oranges", price: 2}]
```

```
products.sort(function(a, b) {  
  if (a.name > b.name) return 1  
  if (a.name < b.name) return -1  
  if (a.name === b.name) {  
    if (a.price > b.price) return 1  
    if (a.price < b.price) return -1  
    return 0  
  }  
})
```

```
console.log(products)
```

```
▼ (4) [{...}, {...}, {...}, {...}] ⓘ  
  ▶ 0: {name: 'Honey', price: 16}  
  ▶ 1: {name: 'Milk', price: 10}  
  ▶ 2: {name: 'Oranges', price: 2}  
  ▶ 3: {name: 'Oranges', price: 4}  
    length: 4  
  ▶ [[Prototype]]: Array(0)
```

```
⏪ undefined
```



Διάσχιση πίνακα με διαγραφή

JavaScript

- Έστω ότι θέλουμε να ελέγξουμε ένα πίνακα για την ύπαρξη ενός στοιχείου και αν υπάρχει να το διαγράψουμε
- Θα χρειαστούμε μία for για να διατρέξουμε τον πίνακα και μία splice() για να διαγράψουμε
- Αν όμως διαγράψουμε καθώς διατρέχουμε τον πίνακα δημιουργείται πρόβλημα γιατί ο πίνακας με τις διαγραφές αναδιοργανώνεται (μειώνεται το length)
- Μία λύση είναι να **διατρέχουμε τον πίνακα από το τέλος προς την αρχή** γιατί η αναδιοργάνωση (reindexing της splice) κατά τη διαγραφή γίνεται από το σημείο της διαγραφής μέχρι το τέλος

```
> let arr = [1, 2, 3, 4, 5]
undefined
> let i = arr.length - 1
undefined
> while (i) {
... if ( arr[i] === 5) arr.splice(i, 1)
... i--
... }
1
> arr
[ 1, 2, 3, 4 ]
```




Διαγραφή με `indexOf` και `splice()`

JavaScript

```
testbed > chapter12 > arrays > <> arrays.html >  htr
6      <meta name="viewport" content="
7      <title>Document</title>
8  </head>
9  <body>
10
11      <script src="https://cdn.jsdelivr.net"
12      <script src="./delete.js"></scr
13  </body>
14  </html>
```



```
JS delete.js  X
testbed > chapter12 > arrays > JS delete.js > ...
1  let arr = [1, 2, 3]
2
3  let itemToRemove = arr.indexOf(2)
4  arr.splice(itemToRemove, 1)
5
6  console.log(arr)
```

- Αν εντοπίσουμε το στοιχείο του πίνακα με `indexOf()` μπορούμε στη συνέχεια να διαγράψουμε με `.splice(index, 1)`



Array Min position (1)

JavaScript

```
1  let arr = [5, 2, 8, 9, 1]
2
3
4  function getMinPosition(arr) {
5      if (!arr || !arr.length) return -1;
6      let minPosition = 0;
7      let min = arr[0]
8
9      for (let i = 1; i < arr.length; i++) {
10         if (arr[i] < min) {
11             min = arr[i]
12             minPosition = i
13         }
14     }
15
16     return minPosition
17 }
18
19 console.log(`MinValue: ${arr[getMinPosition(arr)]}, MinPositionL: ${getMinPosition(arr) + 1}` )
```

- Ελάχιστο στοιχείο πίνακα με τον κλασικό αλγόριθμο



Array Min position (2)

JavaScript

```
testbed > JS arraymin.js > ...  
1  let arr = [1, 2, 3]  
2  let minVal = Math.min(...arr)  
3  let minPos = arr.indexOf(minVal)  
4  
5  console.log(`MinVal: ${minVal} MinPos: ${minPos}`)
```

- Στην JavaScript έχουμε out-of-the-box την `indexOf()`. Μπορούμε με την `Math.min()` και array unpacking με spread operator μπορούμε να βρούμε το ελάχιστο στοιχείο του πίνακα και στη συνέχεια με `indexOf()` να εντοπίσουμε το index



Array Max Value (1)

JavaScript


```
17  
18 function getMaxValue(arr) {  
19     let maxValue = Number.NEGATIVE_INFINITY  
20  
21     for (let item of arr) {  
22         if (item > maxValue) {  
23             maxValue = item  
24         }  
25     }  
26  
27     return maxValue  
28 }  
29
```

- Max value με τον κλασικό τρόπο, με διάσχιση του πίνακα



Array Max Value (2)

JavaScript

```
testbed > JS maxvalue.js >  getMaxValue  
1  const arr = [1, 2, 3, 9, 4, 12, 10]  
2  
3  console.log(getMaxValue(arr))  
4  
5  function getMaxValue(arr) {  
6      return Math.max(...arr)  
7  }
```

- Με `Math.max()` και array desrtructuring με spread operator μπορούμε να πάρουμε το max value



Insert to Array

JavaScript

testbed > js-intro > JS products.js > ...

```
1  const products = []
2
3  let inserted
4
5  try {
6    insert(products, "Honey")
7    console.log("Succesful insert!")
8  } catch (e) {
9    console.log("Product not inserted since already exists")
10 }
11
12 function insert(arr, product) {
13   if (!arr || ! product) return;
14
15   try {
16     if (arr.includes(product)) throw new Error("ProductExists")
17     arr.push(product)
18     return true;
19   } catch (error) {
20     console.log(error.message, error.trace)
21     throw error
22   }
23 }
```

- Η try / catch / finally συντάσσεται όπως και στην Java
- Στην JavaScript όλα τα errors είναι runtime (unchecked) errors
- Η root κλάση είναι η Error από την οποία μπορούμε να κάνουμε extends



Custom Errors

JavaScript

```
25 class ProductAlreadyExistsErrorr extends Error {  
26     constructor(message) {  
27         super(message)  
28         this.name = "ProductAlreadyExists"  
29     }  
30 }
```

- Με την ειδική function constructor αρχικοποιούμε το state του κάθε instance. Μπορούμε και εισάγουμε properties όπως το name (δεν χρειάζεται όπως στην Java να έχουμε ορίσει fields)
- Με super καλούμε τον constructor της parent κλάσης



Prototypal Inheritance (1)

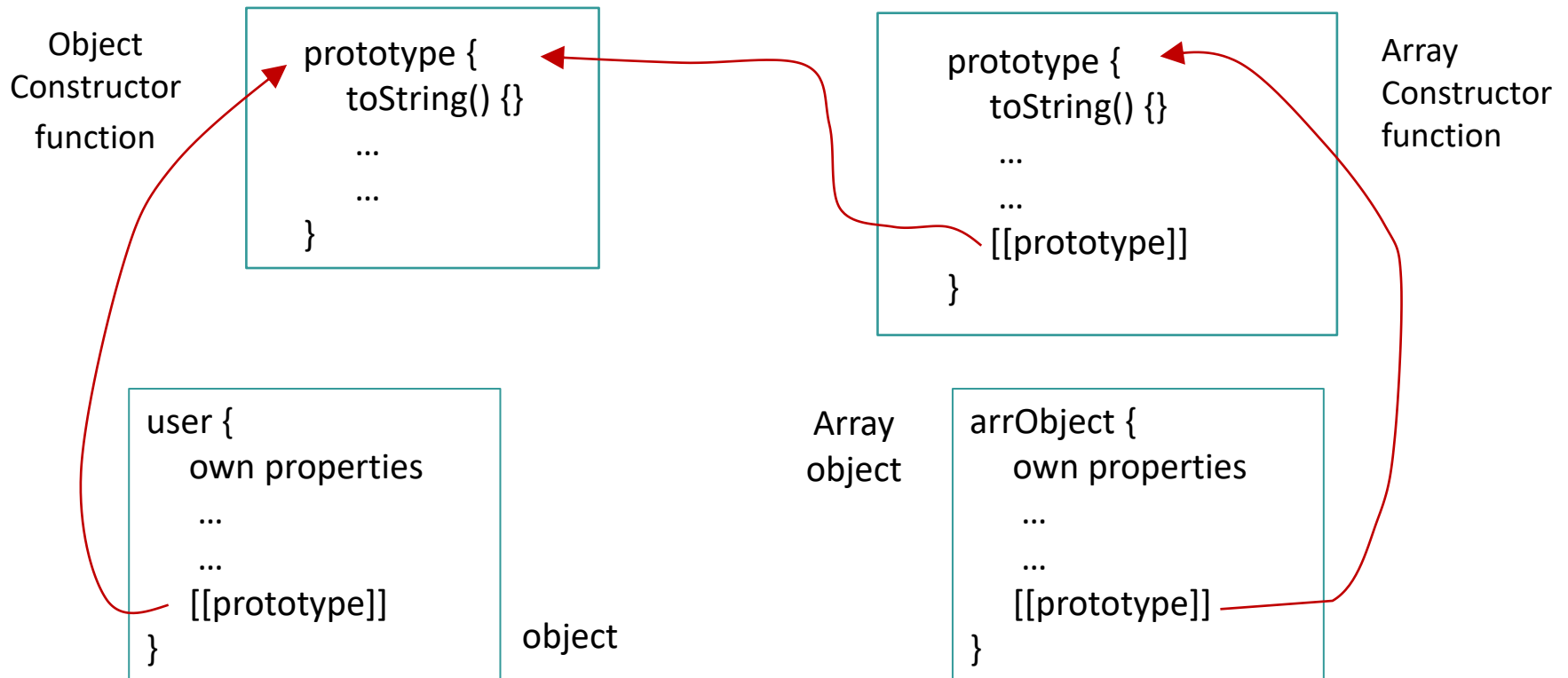
JavaScript

- Κάνοντας `extends` μία κλάση A, μία άλλη parent κλάση B η ιδιότητα `__proto__` (ή `[[prototype]]`) του prototype object της κλάσης A δείχνει στο prototype object της κλάσης B
- Κάθε instance της κλάσης A, έχει επίσης μία ιδιότητα `__proto__` (ή `[[prototype]]`) που δείχνει στο prototype της κλάσης A
- Επομένως τα instances δείχνουν στην prototype property της construction function και 'κληρονομούν' ότι περιέχεται μέσα στο prototype object
- Αν κληθεί από ένα instance μία μέθοδος ή άλλη ιδιότητα που δεν υπάρχει στα own properties, τότε θα ψάξει μέσα στο prototype του Constructor, αν δεν βρεθεί εκεί θα ψάξει στο prototype object του parent constructor, κλπ. δηλαδή στο prototype chain



Prototypal Inheritance (2)

JavaScript



- Το **prototype** είναι ένα object που περιέχουν οι constructor functions. Όταν ορίζουμε μία κλάση, αυτόματα οι μέθοδοι που ορίζουμε εισάγονται σε ένα prototype object της κλάσης και 'κληρονομούνται' στα instances της κλάσης. Όλα τα constructor functions αυτόματα κληρονομούν από το Object type, που είναι το root object στην JavaScript



Shallow Copy

- Όλοι οι τρόποι αντιγραφής στην JavaScript, όπως ο spread operator ή η `Object.assign()` δημιουργούν shallow copies
- Όταν τα στοιχεία ενός πίνακα ή ενός object είναι immutable (πρωταρχικοί τύποι) το shallow copy δεν είναι πρόβλημα, γιατί τα στοιχεία του δεν μπορούν να αλλάξουν
- Αν όμως έχουμε objects ως στοιχεία, τότε θα πρέπει για λόγους συνέπειας να δημιουργούμε deep copies



Deep Copy με JSON

JavaScript

```
testbed > js-intro > JS deepcopy.js > ...
```

```
1  const user = {  
2      id: 1,  
3      username: "alice",  
4      password: "123456"  
5  }  
6  
7  const aliceJsonString = JSON.stringify(user)  
8  const aliceJSClone = JSON.parse(aliceJsonString)
```

- Deep copy με συνδυασμό JSON.parse και JSON.stringify



Lodash

JavaScript

testbed > chapter12 > arrays > <> arrays.html > html > body > script

```
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10
11   <script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.js"></script>
12   <script src="./deep.js"></script>
13 </body>
14 </html>
```

JS deep.js



testbed > chapter12 > arrays > JS deep.js > ...

```
1   const arr = [{id: 1, firstname: 'Alice'}, {id: 2, firstname: 'Bob'}, {id: 3, firstname: 'Bill'}]
2
3   // Deep copy
4   const deepCopy = _.cloneDeep(arr);
5   console.log(deepCopy)
```

- Η Lodash είναι μία βιβλιοθήκη JavaScript που παρέχει utility functions. Το underscore (_) είναι ένα object της Lodash που παρέχει τα static utility functions



Math.random

JavaScript

```
testbed > js-intro > JS random.js > ...
```

```
1 // Generate a random decimal between 0 (inclusive) and 1 (exclusive)
2 const randomDecimal = Math.random();
3
4 // Scale the random decimal to be between 0 and 100
5 const randomNumber = Math.floor(randomDecimal * 101);
6
7 console.log(randomNumber);
```

- Random numbers μπορούμε να πάρουμε με την `Math.random()`
- Με `Math.floor(max-min+1 * random_number) + min` παίρνουμε το range που θέλουμε



Functional Programming

JavaScript

- Για την επεξεργασία των arrays, μπορούμε να αποφύγουμε το boiler plate της while, for, κλπ. iterative programming και να χρησιμοποιήσουμε functional programming, δηλαδή **callbacks**
- Η JavaScript παρέχει out-of-the-box μεθόδους για array manipulation (κληρονομούνται από το Array.prototype)



Array Iterator Methods (1)

JavaScript

- Οι βασικές μέθοδοι που παρέχονται είναι:
 - `forEach()` -- διατρέχει το `array` και εμφανίζει τα στοιχεία του
 - `filter()` -- επιστρέφει `array` με στοιχεία που ικανοποιούν ένα `predicate` (boolean function)
 - `map()` -- επιστρέφει πίνακα με τα 'τροποποιημένα' στοιχεία
 - `reduce()` -- επιστρέφει μία τιμή επί των στοιχείων, π.χ. `sum`
 - `some()` -- επιστρέφει `true` αν έστω ένα στοιχείο ικανοποιεί ένα `predicate`
 - `every()` -- επιστρέφει `true`, αν όλα τα στοιχεία ικανοποιούν ένα `predicate`
 - `find()` -- επιστρέφει το 1^ο στοιχείο που ικανοποιεί ένα `predicate`
 - `findIndex()` -- επιστρέφει το `index` του παραπάνω



Array Iterator Methods (2)

JavaScript

- Αυτές οι μέθοδοι δέχονται ως παράμετρο ένα function (callback function) και εφαρμόζουν αυτό το function σε κάθε στοιχείο του πίνακα
- Συνήθως αυτά τα callbacks γίνονται invoke με τρία arguments: το **value** του κάθε στοιχείου του πίνακα, το **index** του στοιχείου και το ίδιο το **array** αν και μπορούμε να χρησιμοποιήσουμε οποιοδήποτε συνδυασμό των παραπάνω (μόνο το value, ή value και index ή και τις τρεις παραμέτρους)



forEach (1)

- Η **forEach()** είναι μία έτοιμη συνάρτηση της JavaScript που μας διευκολύνει στη διάσχιση πινάκων και γενικά συλλογών, αντί να χρησιμοποιούμε την κλασσική for ή την for .. of
- Περιλαμβάνει μία callback συνάρτηση ως παράμετρο η οποία callback περιλαμβάνει μία υποχρεωτική παράμετρο, που είναι το κάθε στοιχείο του πίνακα



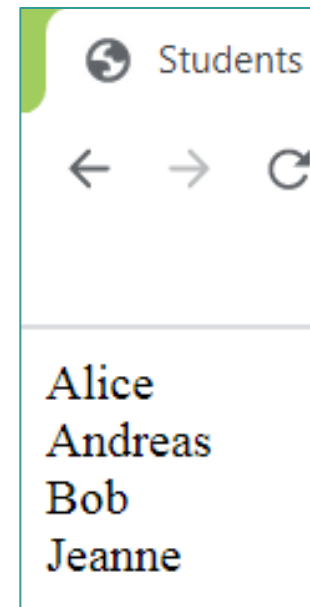
forEach (2)

JavaScript

```
<script>
  let students = ['Alice', 'Andreas', 'Bob', 'Jeanne'];

  students.forEach(function(student) {
    document.write(student + '<br>');
  })

```



- Στην ανώνυμη συνάρτηση που περνάει στην **forEach()** δίνουμε την παράμετρο *student* που αντιστοιχεί στο value κάθε στοιχείο του πίνακα και στο σώμα της συνάρτησης εκτυπώνουμε το στοιχείο



forEach (3)

- Στην **forEach()** μπορούμε να περάσουμε και arrow function ως callback function

```
<script>
  let students = ['Alice', 'Andreas', 'Bob', 'Jeanne'];

  students.forEach(student => document.write(student + '<br>'));

  /* students.forEach(function(student) {
    |   document.write(student + '<br>');
  | }) */
```

Τα δύο snippets (το 2^ο είναι μέσα σε σχόλια) είναι παρόμοια



forEach(4)

JavaScript

- Συνενώνουμε σε ένα τελικό string τα στοιχεία του πίνακα

```
1 let ages = [17, 23, 45, 34, 67, 41, 99]
2 let s = ""
3
4 ages.forEach(function(age) {
5     s += age + " "
6 })
7
8 console.log(s)
```

PROBLEMS

OUTPUT

TERMINAL

DEBUG CONSOLE

```
a8ana@thanassis-pc MINGW64 ~/OneDrive/CodingFactor
$ node ./foreach.js
17 23 45 34 67 41 99
```



Ειδικές Συναρτήσεις Arrays

JavaScript

- Η JavaScript, εκτός από την `forEach()` μας παρέχει κι άλλες έτοιμες συναρτήσεις επεξεργασίας πινάκων ώστε να αποφεύγουμε το boiler-plate, τον επαναλαμβανόμενο δηλαδή κώδικα επεξεργασίας πινάκων με `for .. of` και εντολές επεξεργασίας σε κάθε στοιχείο
- Μερικές έτοιμες συναρτήσεις είναι οι : **`filter()`**, **`map()`** και **`reduce()`**



filter()

JavaScript

```
1 let products = ['Apples', 'Oranges', 'Honey', 'Milk']
2
3 let filtered = products.filter(product => product === "Honey")
4 console.log(filtered)
```

- Όπως λέει και το όνομά της μπορούμε να φιλτράρουμε (*filter*) κάθε τιμή του πίνακα με βάση κάποιο predicate (boolean function) που ορίζουμε στο callback function
- Η **filter()** στο παράδειγμα λαμβάνει ως παράμετρο μία callback function με παράμετρο *product* που αντιστοιχεί σε κάθε στοιχείο του πίνακα



map()

```
testbed > JS map.js > [🔗] mapped
```

```
1 let products = ['Apples', 'Oranges', 'Honey', 'Milk']  
2  
3 let mapped = products.map(product => "Product: " + product)  
4 console.log(mapped)
```

- Όπως λέει και το όνομά της μπορούμε να αντιστοιχίσουμε (**map**) κάθε τιμή του πίνακα σε μία άλλη τιμή. Τον τρόπο αντιστοίχισης τον ορίζουμε μέσα στην **callback function**
- Στο παράδειγμα, η **map()** λαμβάνει ως παράμετρο μία **callback function** με μία παράμετρο έστω *product* που αντιστοιχεί σε κάθε στοιχείο του πίνακα



reduce()

JavaScript

```
1 let nums = [1, 2, 3, 4, 5]
2
3 let sum = nums.reduce((total, val) => total + val, 0)
4 console.log(sum)
```

- Η `reduce()` επιστρέφει ένα συνολικό αποτέλεσμα των στοιχείων του πίνακα σύμφωνα με το σώμα του callback
- Χρειάζεται οπωσδήποτε δύο παραμέτρους, το συνολικό αποτέλεσμα (εδώ *total*) καθώς και το κάθε στοιχείο του πίνακα (εδώ *val*)
- Στο σώμα της περιέχει την πράξη που πρέπει να επιτελεστεί (εδώ `total = total + profit`) καθώς και την αρχική τιμή του `total` (εδώ 0) ως 2^η παράμετρο της `reduce()`
- Αν δεν δώσουμε αρχική τιμή λαμβάνει την πρώτη θέση του πίνακα ως αρχική τιμή



Reduce() (2)

JavaScript

```
testbed > js-intro > JS reduce.js > ...
```

```
1  const arr = [1, 2, 3, 4, 5]
2
3  let sum = arr.reduce((total, val) => total + val)
4  let avg = arr.reduce((total, val) => (total + val)) / arr.length
5
6  console.log(`Sum: ${sum}, Avg: ${avg.toFixed(2)}`)
7
```

- Εδώ η `reduce()` την 1^η φορά επιστρέφει το `sum` , την 2^η φορά διαιρούμε με `arr.length` και παίρνουμε το μέσο όρο (average)



find

JavaScript

```
1 let products = ['Apples', 'Oranges', 'Honey', 'Milk']  
2  
3 let preferred = products.find(product => product.length <= 5)  
4 console.log(preferred)
```

- Επιστρέφει το 1^ο στοιχείο που ικανοποιεί ένα predicate Για παράδειγμα, το Honey είναι το 1^ο στοιχείο που έχει length <= 5
- Η find επιστρέφει (το πρώτο) στοιχείο ενώ αν χρησιμοποιούσαμε filter θα επέστρεφε πίνακα



some

```
testbed > JS some.js > ...
```

```
1 let products = ['Apples', 'Oranges', 'Honey', 'Milk']  
2  
3 let isSomeElemFromA = products.some(product => product.startsWith('A'))  
4 console.log(isSomeElemFromA)
```

- Η `some(callback)` επιστρέφει `true` αν έστω και ένα στοιχείο ικανοποιεί το `callback predicate`
- Για παράδειγμα, το `Apples` ξεκινάει με `A`, επομένως η `some` στον πίνακα `products` επιστρέφει `true`



every

JavaScript

```
1 let products = ['Apples', 'Oranges', 'Honey', 'Milk']
2
3 let hasProducts = products.every(product => product.length >= 1)
4 console.log(hasProducts)
```

- Η `every(callback)` επιστρέφει `true` αν όλα τα στοιχεία ικανοποιούν το `callback` predicate
- Για παράδειγμα, όλα τα `products` έχουν `length >= 1`, επομένως η `every` στον πίνακα `products` επιστρέφει `true`



Ασκήσεις (1)

- Χρησιμοποιήστε .js αρχεία και εκτελέστε με node
1. Δηλώστε μία μεταβλητή 'name' τύπου string και ορίστε το όνομά σας σε αυτή. Εκτυπώστε ένα μήνυμα χρησιμοποιώντας συνένωση όπως "Hello, [όνομα]!"
 2. Δηλώστε δύο μεταβλητές num1 και num2 με αριθμητικές τιμές. Υπολογίστε το άθροισμά τους και εκτυπώστε το αποτέλεσμα
 3. Δηλώστε μια μεταβλητή isTrue με τιμή boolean. Εκτύπωση "Είναι αλήθεια!" αν η isTrue είναι true και "Είναι ψευδές!" σε διαφορετική περίπτωση. Με τριαδικό τελεστή



Ασκήσεις (2)

1. Δημιουργήστε έναν πίνακα που ονομάζεται `cities` με τις τιμές "Athens", "London", "Paris". Προσθέστε "Berlin" στο τέλος του πίνακα. Εκτυπώστε τον ενημερωμένο πίνακα. Διασχίστε τον πίνακα και εκτυπώστε κάθε πόλη
2. Δημιουργήστε ένα object `user` με ιδιότητες, `name`, `age`, `city`. Εκτυπώστε ένα μήνυμα χρησιμοποιώντας αυτές τις ιδιότητες. Προσθέστε μια μέθοδο `hello()` που εκτυπώνει έναν χαιρετισμό χρησιμοποιώντας το `name`. Καλέστε τη μέθοδο `hello()`
3. Γράψτε μια συνάρτηση υπολογισμού του εμβαδόν ενός κύκλου που να παίρνει την ακτίνα ενός κύκλου ως παράμετρο και να επιστρέφει το εμβαδόν (πr^2). Καλέστε τη συνάρτηση με ακτίνα 5



Arrays Ασκήσεις (1)

JavaScript

1. Γράψτε μια συνάρτηση που παίρνει ως παράμετρο έναν πίνακα και επιστρέφει έναν νέο πίνακα που περιέχει μόνο τις μοναδικές τιμές (χωρίς διπλότυπα)
2. Γράψτε μια συνάρτηση που να παίρνει ως παράμετρο έναν `nested` πίνακα και επιστρέφει έναν μονοδιάστατο πίνακα που να περιέχει όλα τα στοιχεία.



Arrays Ασκήσεις (2)

JavaScript

1. Γράψτε μια συνάρτηση που παίρνει έναν πίνακα και ένα μέγεθος υποπίνακα και χωρίζει τον πίνακα σε υποπίνακες του καθορισμένου μεγέθους
2. Γράψτε μια συνάρτηση που παίρνει δύο πίνακες και επιστρέφει έναν νέο πίνακα που περιέχει στοιχεία που είναι κοινά και στους δύο πίνακες



Objects

1. Γράψτε μια συνάρτηση που παίρνει ένα αντικείμενο και έναν πίνακα από keys και επιστρέφει ένα νέο αντικείμενο με μόνο τα ζεύγη key-value των οποίων τα keys βρίσκονται στον πίνακα
2. Γράψτε μια συνάρτηση που παίρνει ως είσοδο ένα αντικείμενο και μια συνάρτηση mapping και επιστρέφει ένα νέο αντικείμενο όπου οι τιμές είναι το αποτέλεσμα της εφαρμογής της συνάρτησης mapping στις αρχικές τιμές
3. Γράψτε μια συνάρτηση που παίρνει ως είσοδο ένα αντικείμενο και μία callback συνάρτηση μετασχηματισμού και μετασχηματίζει το αντικείμενο για κάθε ζεύγος κλειδιού-τιμής (π.χ. μετατρέπει τα keys σε uppercase)