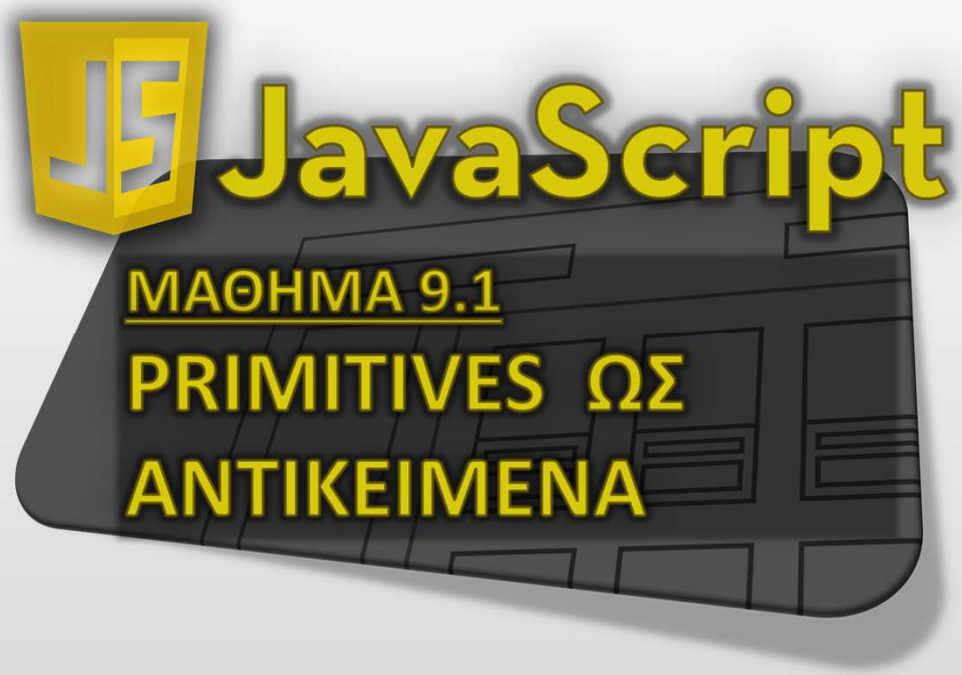

ΠΕΡΙΕΧΟΜΕΝΑ:

1. Αριθμοί
    1. Boxing - Unboxing
    2. Μέθοδοι του Number
2. BigInt
3. Boolean
4. Σύμβολα
    1. Παράδειγμα Χρήσης Συμβόλων

Ανδρέας Γ.
Σμαραγδένιος Χορηγός Μαθήματος

Βασιλική Κ.
Χρυσός Χορηγός Μαθήματος

ΜΑΘΗΜΑ 9.1: Primitives ως Αντικείμενα

1.1. Αριθμοί: Boxing - Unboxing

JavaScript psounis

Υπενθυμίσεις (από μαθήματα 2, 3):

- Ο αριθμός (number) είναι ένα από τα primitives της JS
- Είχαμε δει επίσης την μέθοδο Number(x) που μετατρέπει το όρισμα σε αριθμό.
- Τώρα βλέπουμε πιο αναλυτικά τι είναι το Number.

Το αντικείμενο Number:

- Είναι μια συνάρτηση - κατασκευαστής που περιέχει πολλά χρήσιμα στατικά properties (δεδομένα - μέθοδοι) αλλά και μεθόδους αντικειμένου
- Ως κατασκευαστής:
  - Καλώντας **new Number(x)** ή **Number(x)** κατασκευάζεται ένα νέο αντικείμενο Number (και αν το x δεν είναι αριθμητικό literal θα λάβουν χώρα μετατροπές όπως είδαμε στο μάθημα 3).

Παράδειγμα 1: number

```
let x = new Number(10);
console.log(x);
```

```
Number
  [[Prototype]]: Number
    constructor: ƒ Number()
    toExponential: ƒ toExponential()
    toFixed: ƒ toFixed()
    toLocaleString: ƒ toLocaleString()
    toPrecision: ƒ toPrecision()
    toString: ƒ toString()
    valueOf: ƒ valueOf()
    [[Prototype]]: Object
    [[PrimitiveValue]]: 0
  [[PrimitiveValue]]: 10
```

Boxing - Unboxing:

- Έστω ένα primitive (π.χ. x = 3)
  - Αυτό δεν είναι αντικείμενο Number αλλά primitive (απλή αριθμητική τιμή)
  - Ωστόσο έχουμε πρόσβαση στα properties (π.χ. toExponential()) αν γράψουμε x.toExponential().
  - Αυτό γίνεται γιατί **εσωτερικά γίνεται μετατροπή του primitive σε αντικείμενο Number** και έπειτα καλείται η μέθοδος
  - **(Η διαδικασία αυτή λέγεται boxing)**
- Αντίστροφα:
  - Μπορούμε να χρησιμοποιήσουμε ένα αντικείμενο Number σε έναν υπολογισμό (διότι λαμβάνει χώρα το **unboxing**, κατασκευάζεται primitive και έπειτα λαμβάνει χώρα ο υπολογισμός)

Παράδειγμα 2: boxing

```
let x = 10.112;
console.log(x instanceof Number, typeof x);
console.log(x.toPrecision(4)); //boxing

x = new Number(10.11);
console.log(x instanceof Number, typeof x);
console.log(x + 1); //unboxing;

x = Number(10.11); // as a function (see lesson 3)
console.log(x instanceof Number, typeof x);
```

```
false 'number'
10.11
true 'object'
11.11
false 'number'
>
```

Μέθοδοι Αντικειμένων Number:

- Τα αντικείμενα που έχουν κατασκευαστεί με την Number (ή τα primitives - αριθμοί που γίνονται boxing) έχουν πρόσβαση στις παρακάτω μεθόδους (Number.prototype):

Μέθοδος	Επεξήγηση
toExponential()	Εμφανίζει τον αριθμό σε επιστημονική μορφή
toFixed(n)	Μορφοποιει τον αριθμό με n δεκαδικά ψηφία (με στρογγυλοποίηση)
toPrecision(n)	Μορφοποιεί τον αριθμό με n σημαντικά ψηφία (με στρογγυλοποίηση)
toString()	Μετατρέπει τον αριθμό σε συμβολοσειρά
toLocaleString()	Μετατρέπει τον αριθμό σε συμβολοσειρά (λαμβάνοντας υπόψη το locale του browser)
valueOf()	Επιστρέφει την τιμή - αριθμό

Παράδειγμα 3: number

```
let x = 10.115;
console.log(x, x.toExponential());
console.log(x.toString(), x.toLocaleString());
console.log(x.toFixed(2), x.toPrecision(3));
```

```
10.115 '1.0115e+1'
10.115 10.115
10.12 10.1
>
```

Σημειώσεις:

- Οι μέθοδοι αυτοί είναι μέλη του πρωτοτύπου Number.prototype
- Δεν μπορούμε να εφαρμόσουμε τις μεθόδους σε literals (δεν λαμβάνει χώρα boxing-unboxing):
  - π.χ. δεν είναι συντακτικά έγκυρο: 12.11.toPrecision(3)

Στατικές Μέθοδοι της Number

- Το αντικείμενο - κατασκευαστής Number ορίζει τις στατικές μεθόδους:

Μέθοδος	Επεξήγηση
isNaN(x)	true, αν ο αριθμός x είναι NaN
isFinite(x)	true, αν ο αριθμός είναι πεπερασμένος
isInteger(x)	true, αν ο αριθμός είναι ακέραιος
parseInt(x, radix)	Μετατρέπει τη συμβολοσειρά x σε ακέραιο με βάση το αριθμητικό σύστημα radix
parseFloat(x, radix)	Μετατρέπει τη συμβολοσειρά x σε πραγματικό με βάση το αριθμητικό σύστημα radix

Στατικά Properties Δεδομένων:

- EPSILON, MAX_VALUE, MIN_VALUE, NaN, POSITITIVE_INFINITY, NEGATIVE_INFINITY, MIN_SAFE_INTEGER, MAX_SAFE_INTEGER

Παράδειγμα 4: number_static

```
let N = Number;
console.log(N.EPSILON); //min interval: consecutive numbers
console.log(N.MIN_SAFE_INTEGER, N.MAX_SAFE_INTEGER);
console.log(N.MIN_VALUE, N.MAX_VALUE);
console.log(N.NEGATIVE_INFINITY, N.POSITIVE_INFINITY);

console.log(N.isNaN(NaN), NaN===NaN);
console.log(N.isFinite(N.MAX_VALUE));
console.log(N.parseInt("5"), N.parseInt("11", 2));
```

ΜΑΘΗΜΑ 9.1: Primitives ως Αντικείμενα

2. BigInt

JavaScript psounis

Το αντικείμενο BigInt:

- Ενθυλακώνει με λειτουργικότητα primitives τύπου bigint
- Για την κατασκευή χρησιμοποιούμε τη συναρτηση (προσοχή χωρίς new, δεν είναι συνάρτηση - κατασκευαστής) :
  - Καλώντας **BigInt(x)** κατασκευάζεται ένα νέο αντικείμενο BigInt (και αν το x δεν είναι αριθμητικό literal θα λάβουν χώρα μετατροπές όπως είδαμε στο μάθημα 3).
- Στατικές Μέθοδοι:

Μέθοδος	Επεξήγηση
asIntN(b, n)	Επιστρέφει έναν αριθμό (bigint) κρατώντας τα b λιγότερο σημαντικά bit του αριθμού n (σε απεικόνιση συμπληρώματος ως προς 2)
asUIntN(b, n)	Επιστρέφει έναν αριθμό (bigint) κρατώντας τα b λιγότερο σημαντικά bit του αριθμού n (ως μη προσημασμένο ακέραιο)

- Μέθοδοι Αντικειμένων:

Μέθοδος	Επεξήγηση
toString([n])	Επιστρέφει τον αριθμό ως συμβολοσειρά. n: Εφόσον οριστεί, εκφράζει την βάση του αριθμητικού συστήματος στο οποίο θα εκφραστεί ο αριθμός
toLocaleString (locale)	Επιστρέφει τον αριθμό στην μορφοποίηση της τοπικής προσαρμογής (π.χ. loc = “en-EN”, “el-GR”)
valueOf()	Επιστρέφει την τιμή (υπερβαίνει την valueOf του Object)

Παράδειγμα 5: BigInt

```
let x = BigInt(10);
console.log(x, typeof x);

// let x = new BigInt(10); // doesn't work
console.log(Object(10n));

console.log(BigInt.asIntN(2, 2n));
console.log(BigInt.asUintN(6, 2n));

console.log(x, x.toString());
console.log(x.toString(2));
console.log(x.toString(8));

x = 101.33;
console.log(x.toLocaleString("el-GR"));
console.log(x.toLocaleString("en-EN"));
console.log(x.toLocaleString("de-DE"));
```

```
▼ BigInt {10n}
  ▼ [[Prototype]]: BigInt
    ▶ constructor: ƒ BigInt()
    ▶ toLocaleString: ƒ toLocaleString()
    ▶ toString: ƒ toString()
    ▶ valueOf: ƒ valueOf()
      Symbol(Symbol.toStringTag): "BigInt"
    ▶ [[Prototype]]: Object
    [[PrimitiveValue]]: 10n
```

ΜΑΘΗΜΑ 9.1: Primitives ως Αντικείμενα

3. Boolean

JavaScript psounis

Το αντικείμενο Boolean:

- Αντίστοιχα με το Number, το αντικείμενο Boolean είναι αντικείμενο - περιτυλιχτής boolean primitives.
- Ως κατασκευαστής:
  - Καλώντας **new Boolean(x)** κατασκευάζεται ένα νέο αντικείμενο Boolean (και αν το x δεν είναι boolean literal θα λάβουν χώρα μετατροπές όπως είδαμε στο μάθημα 3).

Παράδειγμα 6: boolean

```
let x = new Boolean(10);
console.log(x, x===true, x==true);
```

```
▸ Boolean {true} false true
```

Υπενθύμιση (από μάθημα 2):

- Σε λογικους ελέγχους (π.χ. στη συνθήκη της if) οι τιμές:
  - **falsy τιμές:** 0, undefined, null, NaN και η κενή συμβολοσειρά
  - **truthy τιμές:** Όλες οι υπόλοιπες

Παρατήρηση:

- Σπανίως (αν όχι ποτέ) δεν θα χρησιμοποιήσουμε αντικείμενα Boolean
- (Προχωρημένοι προγραμματιστές ισχυρίζονται ότι κάποιος μπορεί να κάνει κόλπα μέσω αντικειμένων Boolean)

Μέθοδοι αντικειμένων Boolean

- Περιέχονται στο Boolean.prototype

Μέθοδος	Επεξήγηση
toString()	Μετατρέπει τον Boolean σε λογική μεταβλητή
valueOf()	Επιστρέφει την τιμή - primitive

- (Δεν υπάρχουν στατικά μέλη)

Παράδειγμα 7: boolean_methods

```
let x = true;
console.log(x.valueOf(), x.toString());
console.log(Boolean(Number(5)).valueOf().toString())
```

Σημείωση για τα primitives: null, undefined

- Στο μάθημα 2 είδαμε:
  - Το null είναι λέξη - κλειδί που αναπαρίσταται με ένα αντικείμενο που έχει μοναδική τιμή null
  - Το undefined είναι μία καθολική σταθερά
- Και τα δύο primitives δεν έχουν κάποιο αντικείμενο - περιτυλιχτή (σαν τον Number και το Boolean)

ΜΑΘΗΜΑ 9.1: Primitives ως Αντικείμενα

4. Σύμβολα

JavaScript psounis

Πρόβλημα:

- Η JS όπως είδαμε είχε πολλά προβλήματα ασυνέπειας στις αρχικές της εκδόσεις.
- Σε επόμενες εκδόσεις ήθελαν να προσθέσουν λειτουργικότητα (π.χ. έξτρα μεθόδους στο Object)
- Αυτό όμως θα «έσπαγε» παλιότερο κώδικα προγραμματιστών (που μπορεί να είχαν ήδη ορίσει μεθόδους με ίδια ονόματα)

Λύση:

- Ένας νέος τύπος δεδομένων - primitive: **Το σύμβολο, που παράγει μοναδικά αντικείμενα**, που μπορούν να χρησιμοποιηθούν ως κλειδιά των properties αντικειμένων.

Κατασκευή Συμβόλου:

- Προσοχή ότι δεν είναι συνάρτηση - κατασκευαστής (όχι με new):

Συνάρτηση	Επεξήγηση
Symbol()	Επιστρέφει μοναδικό σύμβολο
Symbol(description)	Προσδιορίζει το σύμβολο με τη συμβολοσειρά description (μόνο για debugging)

Παράδειγμα 8: symbol

```
let s = Symbol();
console.log(s, Object(s));
let x = Symbol("symbol1");
let y = Symbol("symbol1");
console.log(x==y, Symbol()==Symbol());
```

Παρατηρήσεις:

- Δεν υπάρχει literal για σύμβολα.
- Περιέχει αρκετές μεθόδους που είναι αρκετά προχωρημένες σε δυσκολία.
- Ενδιαφέρον παρουσιάζει η στατική μέθοδος:

Στατ. Μέθοδος	Επεξήγηση
for(str)	Επιστρέφει κοινό σύμβολο (δηλαδή, σε αντίθεση με τη συνάρτηση Symbol(), επιστρέφεται πάντα το ίδιο σύμβολο)

Παράδειγμα 9: BigInt

```
let s = Symbol.for("shared");
let t = Symbol.for("shared");
console.log(s == t);

let object1 = {
  [Symbol.for("shared")]: 1
}
let object2 = {
  [Symbol.for("shared")]: 1
}
console.log(object1.s == object2.s);
console.log(object1);
```

```
true
true
▼ {Symbol(shared): 1}
    Symbol(shared): 1
  ▶ [[Prototype]]: Object
```

ΜΑΘΗΜΑ 9.1: Primitives ως Αντικείμενα 4.1. Παράδειγμα Χρήσης Συμβόλου

JavaScript psounis

Σημείωση:

- Θα ασχοληθούμε εκτενώς με τους iterators/generators σε έπόμενα μαθήματα.
- Βλέπουμε εδώ ένα preview αυτών των μαθημάτων, ώστε να εξοικειωθούμε με την ιδέα της χρήσης των συμβόλων για χρήσιμες επεκτάσεις της γλώσσας.

Χρήση Συμβόλων σε Επαναληπτές (Iterators):

- Ορίζουμε επί ενός αντικειμένου, έναν επαναληπτή (iterator) ώστε το αντικείμενο να δέχεται επανάληψη (iterable)
  - Χρησιμοποιείται από την for…of ώστε να επιστρέφει τιμές (με τη σειρά που θέλουμε).

Παρατηρήσεις

- Symbol.iterator: Στατικό μέλος - σύμβολο που χρησιμοποιείται για τον ορισμό iterator σε ένα αντικείμενο.
- yield: Αντικαθιστά τη return (Επιστροφή Τιμής σε Συναρτήσεις - generators)
- *function():
  - Συνάρτηση - generator: Γεννά τιμές. Βασική ιδιότητα: Συνεχίζει την εκτέλεση της από εκεί που σταμάτησε την προηγούμενη φορά (έτσι π.χ. εδώ, στην πρώτη κλήση επιστρέφει το this.x και στη δεύτερη το this.y)
- Η συνάρτηση - generator καλείται αυτόματα όταν γίνεται επανάληψη επί του αντικειμένου με το for..of loop.

Παράδειγμα 10: symbol_iterator

```
let object = {
  x: 1,
  y: 2,

  *[Symbol.iterator]() {
    yield this.x;
    yield this.y;
  }
}

for (let item of object)
  console.log(item);
```

Άσκηση 1:

- Επεκτείνοντας την άσκηση 1 του μαθήματος 8.5, ορίστε στην κλάση Point έναν Επαναληπτή, ο οποίος να επιστρέφει τις συντεταγμένες του σημείου.