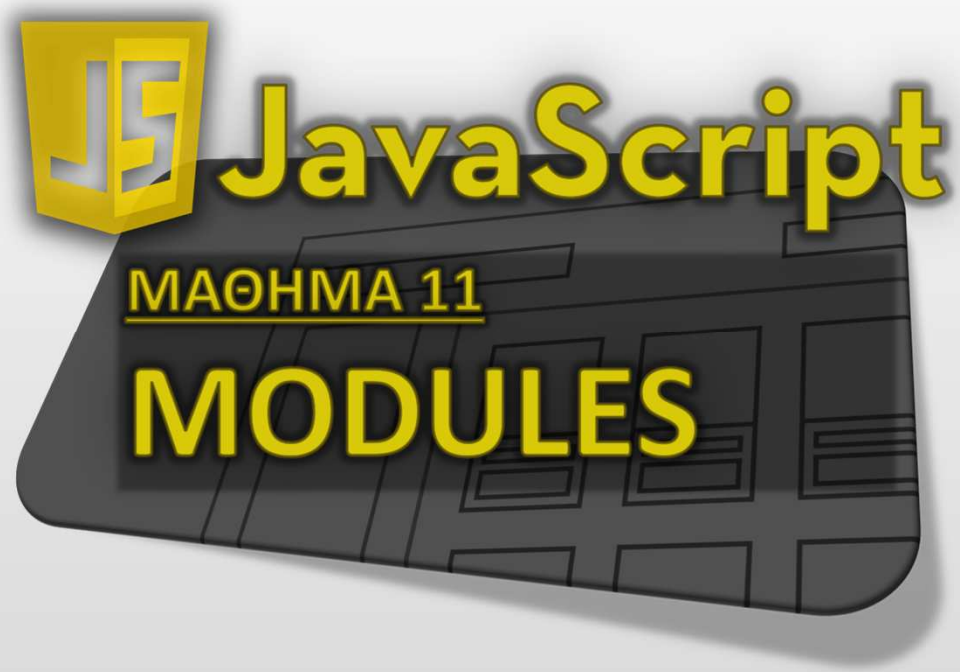


ΠΕΡΙΕΧΟΜΕΝΑ:

Υπενθυμίσεις από μάθημα 3:

- Μέχρι τώρα έχουμε δει πως ενσωματώνουμε κώδικα JS στην εφαρμογή μας με δύο τρόπους:
  - Ως inline scripts
  - Με το tag <script>
- Τα ονόματα που ορίζονται σε αυτά, αποθηκεύονται:
  - Στο καθολικό περιβάλλον εκτέλεσης (δηλώσεις let/const)
  - Στο αντικείμενο window που έχει αποθηκευτεί στο καθολικό περιβάλλον εκτέλεσης (δηλώσεις var)
- Με βάση τα παραπάνω:
  - Στο σώμα είτε inline, είτε <script> που τρέχουν σε ένα tab του Chrome, βλέπουμε όλα τα ονόματα που έχουν δηλωθεί σε προηγούμενα βήματα.

Χώρος Ονομάτων (Namespace)

- Στον προγραμματισμό, βαφτίζουμε ως χώρο ονομάτων (namespace), μία περιοχή που αποθηκεύουμε ονόματα
- Πρακτικά, στην JS είχαμε έναν **κοινό χώρο ονομάτων** για τα πάντα (το καθολικό περιβάλλον εκτέλεσης):
  - **Σημαντικό πρόβλημα:** Αν φανταστούμε ότι μία εφαρμογή μπορεί να ενσωματώνει πολλές βιβλιοθήκες, θα προκαλούνται **συγκρούσεις ονομάτων**. Λύσεις:
    - < ES6: IIFE
    - >= ES6: Modules

Παράδειγμα 1: reminder/reminder.html

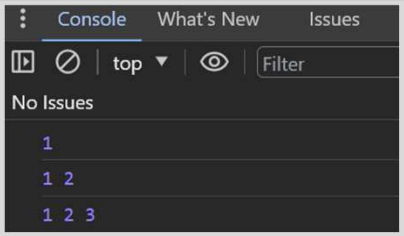
```
...
<head>
 ...
 <script src="script1.js"></script>
</head>
<body>
 <script>
  let b=2;
  console.log(a,b);
 </script>
 <script src="script2.js"></script>
</body>
</html>
```

script1.js

```
let a=1;
console.log(a);
```

script2.js

```
let c=3;
console.log(a,b,c);
```



ΜΑΘΗΜΑ 11: Modules

2.1. Named Exports

JavaScript psounis

- Όταν τα προγράμματα μας "ξεφεύγουν" σε μέγεθος:
  - Οργανώνουμε τον κώδικα σε διαφορετικά αρχεία, τα οποία τα λέμε modules.
  - **Module είναι ένα αρχείο κώδικα js, το οποίο το ενσωματώνουμε (import) σε άλλο αρχείο κώδικα** ώστε:
    - Να αποκρύπτουμε λεπτομέρειες της υλοποίησης
    - Να μπορούμε να το ξαναχρησιμοποιούμε σε άλλα προγράμματα
    - Να τα διαμοιραζόμαστε με άλλους προγραμματιστές.
    - Να μην έχουμε συγκρούσεις ονομάτων

Στο αρχείο .js που θα κάνει export ονόματα

- Βάζουμε τη λέξη - κλειδί **export** μπροστά από το όνομα που θέλουμε να κάνουμε εξαγωγή σε άλλο αρχείο.
- Λέμε ότι κάνουμε ένα **export με βάση το όνομα (named export)**

Παράδειγμα 2: named/lib.js

```
export let a = 1;
let b = 2;

export function f() {
  console.log("f");
}
function g() {
  console.log("g");
}
```

Εισαγωγή κάποιων ονομάτων

Ο κώδικας js που θα ενσωματώσει ονόματα από άλλο αρχείο js:

- Σύνταξη: **import {name1, name2, …} from 'file'**
- (τα name1, name2, … πρέπει να έχουν γίνει export από το αρχείο file).

Παράδειγμα 3: named/program.js

```
import {a, f} from "./lib.js";

console.log(a);
console.log(f);
```

Παρατηρήσεις:

- Import γίνεται μόνο σε άλλο module. Δεν μπορούμε να κάνουμε import σε inline script.
- Χρησιμοποιούμε σχετικό μονοπάτι στον καθορισμό του αρχείου - module από το οποίο κάνουμε Import
- Για να ενσωματώσουμε ένα module στον κώδικά HTML απαιτείται να καθορίσουμε ότι αυτό είναι module:
  - Στο στοιχείο **script** θέτουμε το χαρακτηριστικό **type="module"**

```
<head>
  …
  <script src="program.js" type="module"></script>
</head>
```

Εισαγωγή όλων των ονομάτων που γίνονται export από module

- Σύνταξη: **import * as moduleName from 'file'**
  - moduleName: Βαφτίζουμε με όνομα της αρεσκείας μας το module (υποχρεωτικό)
  - Η πρόσβαση στα ονόματα γίνεται ως moduleName.name

Παράδειγμα 4: import_all

lib.js:

```
export let a = 1;
let b = 2;

export function f() {
   console.log("f");
}
function g() {
   console.log("g");
}
```

program.js:

```
import * as myLib from "./lib.js";

console.log(myLib.a);
console.log(myLib.f);
```

Εισαγωγή κάποιων ονομάτων με συνώνυμα:

Μπορούμε να ορίσουμε συνώνυμο για κάποιο όνομα που εισάγουμε ως:

- Σύνταξη: **import {name as alias, …} from 'file'**

Παράδειγμα 5: named/program.js

```
import {a, f as func} from "./lib.js";

console.log(a);
console.log(func);
```

Άσκηση 1:

- Κατασκευάστε ένα module με όνομα shapes.js το οποίο περιέχει δύο συναρτήσεις:
  - triangle(base, height), υπολογίζει και επιστρέφει το εμβαδόν τριγώνου (βάση * ύψος / 2)
  - circle(radius), υπολογίζει και επιστρέφει το εμβαδόν κύκλου (2π*ακτίνα)
- 'Επειτα κατασκευάστε ένα module με όνομα index.js που καλεί τις δύο παραπάνω συναρτήσεις.

Default Export

- Το χρησιμοποιούμε (συνήθως) όταν θέλουμε ακριβώς ένα όνομα να εξάγεται από ένα module.
- Γράφουμε μπροστά από το όνομα που θα εξάγουμε:
  - **export default …**
- Διαφοροποιείται η εντολή εισαγωγής σε άλλο module:
  - **import name [as alias] from 'file'**
  - (Δεν βάζουμε άγκιστρα στο name)

Παράδειγμα 6: export_default

lib.js:

```
…
export default function g() {
   console.log("g");
}
```

program.js:

```
import g from "./lib.js";
console.log(g);
```

Παρατήρηση:

- Μπορούμε να αλλάξουμε το όνομα κατά την εισαγωγή, π.χ. στο παραπάνω παράδειγμα:

```
import func from "./lib.js";
console.log(func);
```

Παρατηρήσεις:

- Μπορούμε να έχουμε ταυτόχρονα από ένα module:
  - Ακριβώς ένα default export,
  - πολλά named exports
- Η εντολή για το default export μπορεί να έπεται του ορισμού του ονόματος.

Παράδειγμα 7: export_and_named

lib.js:

```
let a = 1;
export let b = 2;

export function f() {
   console.log("f");
}

function g() {
   console.log("g");
}
export default g;
```

program.js:

```
import theDefault, {b, f as theNamed} from "./lib.js";
theNamed();
theDefault();
console.log(b);
```

Export & Import

- Εάν θέλουμε να κάνουμε export ένα όνομα που εισάγουμε από ένα άλλο module, αντί για να γράψουμε:
  - **import {name} from 'file';**
  - **export name;**
- Μπορούμε να γράψουμε:
  - **export {name} from 'file';**

Παράδειγμα 8: export_from_import

lib.js:

```
export function f() {
  console.log("f");
}
```

lib2.js:

```
export {f} from "./lib.js";
```

program.js:

```
import {f} from "./lib2.js";

console.log(f);
```

index.html

```
<head>
 ...
 <script src="program.js" type="module"></script>
</head>
```

Δύο σημαντικές επισημάνσεις:

- Από τα παραπάνω έχει γίνει κατανοητό:
  - Ο κώδικας του module τρέχει κάθε φορά που το κάνουμε import.
- Όπως είπαμε, προσοχή ότι οι εντολές import/export ορίζουν ότι ένα αρχείο js είναι module και απαιτείται στο στοιχείο script.

Συγκρούσεις Ονομάτων:

- Ατιμετωπίζουμε συγκρούσεις ονομάτων με δύο τρόπους:
  - **είτε κάνουμε import ολόκληρα τα modules**
  - **είτε κάνουμε μετονομασία σε κάποιο συγκρουόμενο όνομα**

Παράδειγμα 10: name_conflict

lib.js:

```
export let x = 1;
```

lib2.js:

```
export let x = 1;
```

program.js:

```
import * as lib from "./lib.js";
import * as lib2 from "./lib2.js";
console.log(lib.x, lib2.x);
```

Παράδειγμα 3: name_conflict2

program.js:

```
import {x} from "./lib.js";
import {x as xLib2} from "./lib2.js";

console.log(x, xLib2);
```

IIFE (Immediately Invoked Function Expression):

- (βλ. και μάθημα 7)
- Ένας δημοφιλής τρόπος για «τεχνητά» modules την < ES6 εποχή.
- Κάθε IIFE έχει τη δική του εμβέλεια, οπότε αξιοποιώντας τις ιδιότητες των closures, επιτυγχάνεται αντίστοιχη λειτουργικότητα.

Παράδειγμα 11: iife

lib.js

```
let lib = (function() {
   let x = 1;
   return {x}; // export the object
})();
```

lib2.js

```
let lib2 = (function(lib) { // import lib
   let x = 2;
   return {x}; // export x
})(lib);
```

program.html:

```
console.log(lib.x, lib2.x);
```