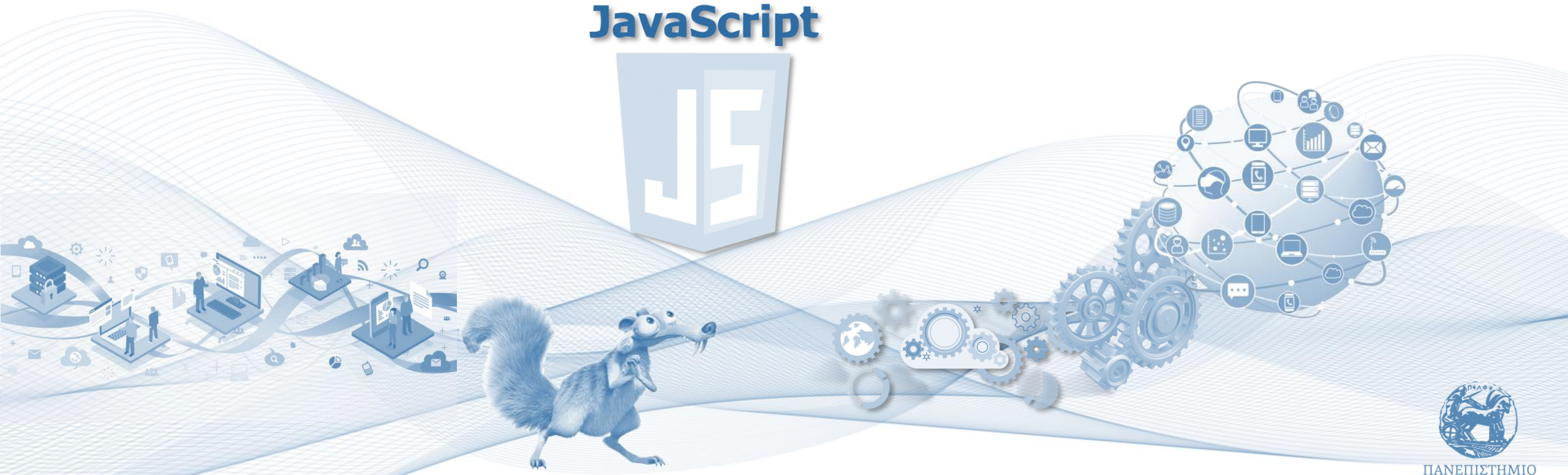
Τμήμα Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ

ΤΕΧΝΙΚΕΣ ΠΡΟΓΡΑΜΜΑΤΙΜΟΥ ΥΠΟΛΟΓΙΣΤΩΝ

Διάλεξη V

JavaScript από πιο κοντά: Τα Αντικείμενα και οι Έλεγχοι

Γιάννης Τζήμας, Καθηγητής

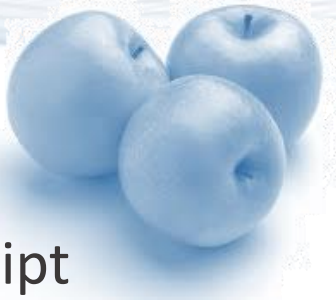


ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ
UNIVERSITY of the PELOPONNESE

Τι θα δούμε σήμερα…

- Σε αυτό το μάθημα θα δούμε τα ακόλουθα:
  - Την έννοια και χρήση των Αντικειμένων– Objects,
  - Τρόπους για να κάνουμε ελέγχους σε ένα πρόγραμμα.

Αντικείμενα – Objects



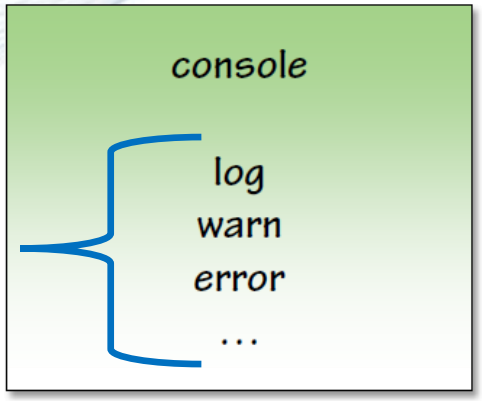
- Υπάρχουν πολλές σύγχρονες γλώσσες προγραμματισμού που υποστηρίζουν την έννοια των **αντικειμένων**, μεταξύ των οποίων και η JavaScript. Η JavaScript διαθέτει έναν αριθμό τεχνικών για τη δημιουργία αντικειμένων. Θα υιοθετήσουμε μία εύκολη προσέγγιση, ώστε να μάθουμε μερικές από τις βασικές αρχές εργασίας με αντικείμενα.

- **Μας επιτρέπουν να ομαδοποιούμε συσχετιζόμενα δεδομένα και συμπεριφορά σε ένα «δοχείο».**
  - **Αυτό το δοχείο το ονομάζουμε αντικείμενο.**

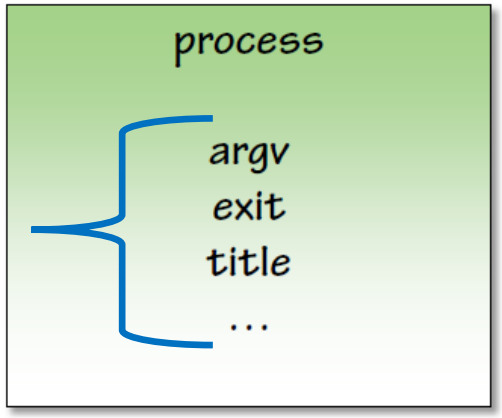
- Τα αντικείμενα δεν υποστηρίζονται από όλες τις γλώσσες προγραμματισμού (σε πολλές υποστηρίζονται), και σχεδόν όλες διαθέτουν κάποια έννοια ομαδοποίησης, ή οργάνωσης κομματιών προγράμματος, ώστε να το κάνουν πιο εύκολα διαχειρίσιμο.
  - Αυτός είναι και ένας λόγος χρησιμοποίησης των αντικειμένων, η **οργάνωση**.

Τα έχουμε ξαναδεί μέχρι τώρα...

Μέλη – Members της console. Όλες παρουσιάζουν κείμενο στο χρήστη.



Ιδιότητες/Γνωρίσματα Properties/Attributes του αντικειμένου.




```
console.log("Hello");
```

Για να έχω πρόσβαση στο μέλος log της console χρησιμοποιώ την τελεία.

```
console.log(process.title);
```

- Αν δε μπορούσαμε να οργανώσουμε τις συναρτήσεις με τη βοήθεια των αντικειμένων θα ήταν δύσκολο να δουλέψουμε σε μεγάλα προγράμματα.

- Τα αντικείμενα μας βοηθούν να **σκεπτόμαστε με όρους πραγματικών αντικειμένων**, με τα οποία μπορούμε να αλληλεπιδράσουμε. Και καθώς τα αντικείμενα έχουν όνομα, το μυαλό μας μπορεί ποιο εύκολα να **κατανοήσει** τη δομή και το τι συμβαίνει στο πρόγραμμα.
- Αν είχαμε ένα αντικείμενο για ένα καλάθι αγορών, θα είχε μέλη με τα οποία θα μπορούσαμε να προσθέσουμε προϊόντα για αγορά.
- Το αντικείμενο καλάθι αγορών θα παρακολουθούσε το ποια προϊόντα προστέθηκαν. Αυτό είναι ένα ακόμα σημαντικό χαρακτηριστικό των αντικειμένων - εκτός από μέλη, έχουν και **κατάσταση** που στην πραγματικότητα είναι τα **δεδομένα** τα οποία αποθηκεύει το αντικείμενο.

Methods & Properties

Object	Properties	Methods
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white	car.start() car.drive() car.brake()

Αντικείμενα στο REPL

```
Hello there!
C:\examples>node
> 2>=3
false
> 2>3
false
> 2+2
4
> "hello"
'hello'
> message = "Hello there!"
'Hello there!'
> message
'Hello there!'
> console
Console {
  log: [Function: bound ],
  info: [Function: bound ],
  warn: [Function: bound ],
  error: [Function: bound ],
  dir: [Function: bound ],
  time: [Function: bound ],
  timeEnd: [Function: bound ],
  trace: [Function: bound trace],
  assert: [Function: bound ],
  Console: [Function: Console] }
> console.log
[Function: bound ]
> process.argv
[ 'C:\\Program Files\\nodejs\\node.exe' ]
> process.argv[0]
'C:\\Program Files\\nodejs\\node.exe'
> process.title
'Command Prompt - node'
> process.title = "Hello!"
'Hello!'
> process.title = message
'Hello there!'
```

Read

Set

```
> process
process {
  title: 'Hello there!',
  version: 'v6.11.5',
  moduleLoadList:
   [ 'Binding contextify',
     'Binding natives',
     'Binding config',
     'NativeModule events',
     'NativeModule util',
     'Binding uv',
     'NativeModule buffer',
     'Binding buffer',
     'Binding util',
     'NativeModu
     'NativeModu
     'Binding ti
```

```
      v8_inspector: true,
      v8_no_strict_aliasing: 1,
      v8_optimized_debug: 0,
      v8_random_seed: 0,
      v8_use_snapshot: true,
      want_separate_host_toolset: 0 } },
  emitWarning: [Function],
  nextTick: [Function: nextTick],
  _tickCallback: [Function: _tickDomainCallback],
  _tickDomainCallback: [Function: _tickDomainCallback],
  stdout: [Getter],
  stderr: [Getter],
  stdin: [Getter],
  openStdin: [Function],
  exit: [Function],
  kill: [Function],
  argv0: 'node' }
>
```

Ας ορίσουμε τώρα τα δικά μας αντικείμενα…

```
var die = {

};
```

Object Literal

Μπορείτε να δημιουργήσετε αντικείμενα κατ' απαίτηση.

```
var die = {

};

console.log(die);
```

```
C:\examples>node program_0.js
{}
```

Property

```
var die = {
    size:4
};

console.log(die);
```

```
C:\examples>node program_0.js
{ size: 4 }
```

```
var die = {
    size:4
};

console.log(die);
console.log(die.size);

die.size = 10;

console.log(die.size);
```

```
C:\examples>node program_0.js
{ size: 4 }
4
10
```

```
var die = {
    size:4
};

console.log(die);
console.log(die.size);
```

```
C:\examples>node program_0.js
{ size: 4 }
4
```

- Δοκιμάστε να κάνετε console.log(die) για να δείτε την τιμή του size εντός του αντικειμένου.

Ας ορίσουμε τώρα τα δικά μας αντικείμενα… Επιπλέον Μέλη

```
var die = {
    size: 4,
    count: 1,
    roll: function(dieSize){
        var result = Math.ceil(dieSize * Math.random());
        return result;
    }
};

console.log(die);
console.log(die.roll(6));
```

```
C:\examples>node program_0.js
{ size: 4, count: 1, roll: [Function: roll] }
2

C:\examples>node program_0.js
{ size: 4, count: 1, roll: [Function: roll] }
6

C:\examples>node program_0.js
{ size: 4, count: 1, roll: [Function: roll] }
6
```

- Παρατηρείστε:
  - Τα properties διαχωρίζονται με κόμμα.
  - Χρησιμοποιούμε άνω κάτω τελεία «:» όταν δίνουμε τιμή σε property, ή ορίζουμε μία συνάρτηση.

Ας δώσουμε μερικές ευθύνες στο αντικείμενό μας…

- Τα αντικείμενα συχνά έχουν ένα σκοπό.
- Στην περίπτωσή μας το ζάρι θα μπορούσε να έχει κάποιες ευθύνες, όπως το να θυμάται το πόσες πλευρές έχει, ή το πόσες φορές έχει ριχτεί.

```
var die = {
    size: 6,
    totalRolls: 0,
    roll: function(){
        var result = Math.ceil(this.size * Math.random());
        this.totalRolls += 1;
        return result;
    }
};


die.size = 10;
console.log(die.roll());
console.log(die.roll());
console.log(die.roll());
console.log("Total rolls " + die.totalRolls);

console.log(die);
```

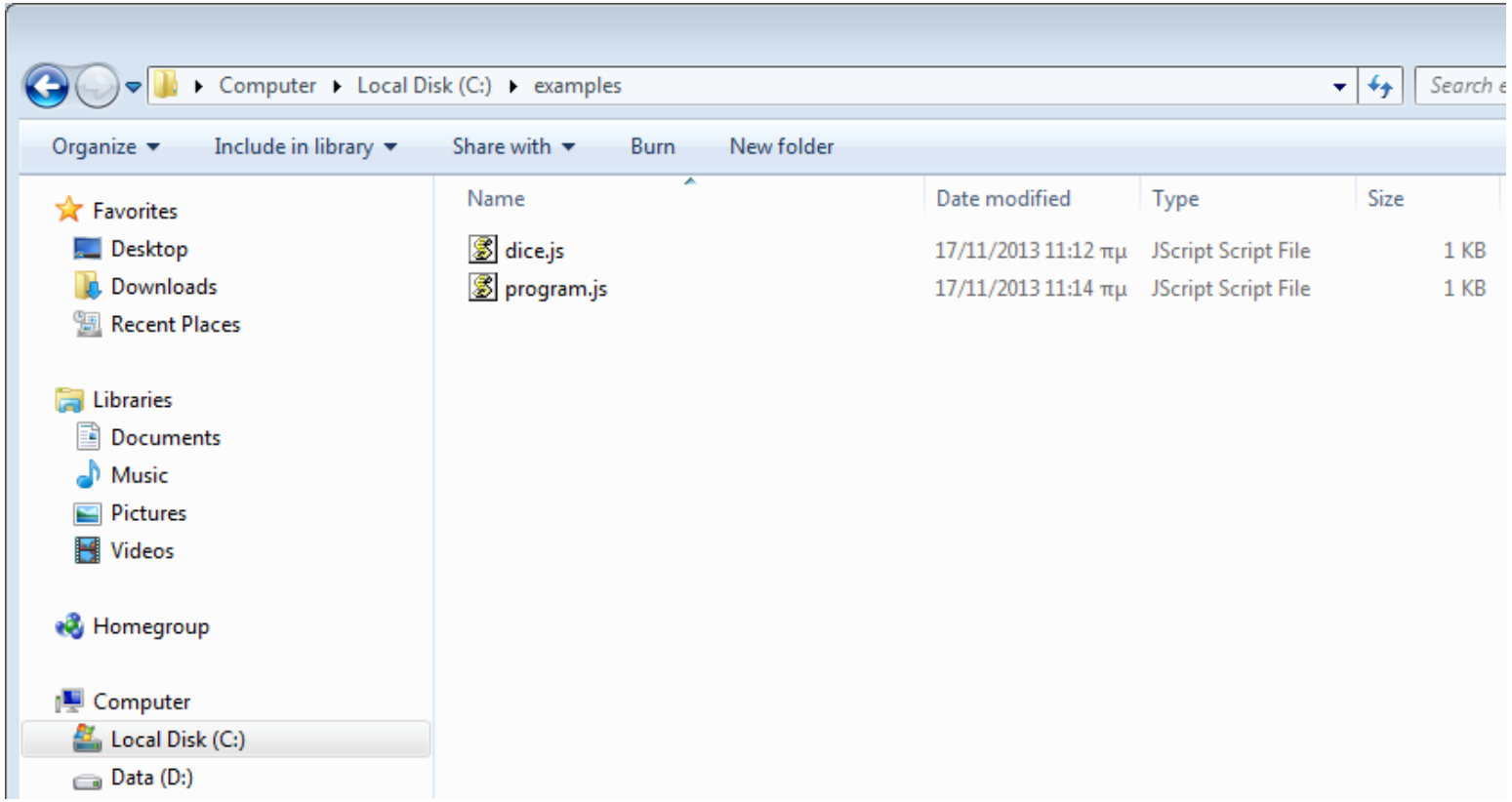
```
C:\examples>node program_0.js
8
6
10
Total rolls 3
{ size: 10, totalRolls: 3, roll: [Function: roll] }
```

Προσοχή!
Το **`this`** είναι δεσμευμένη λέξη και με βοηθάει να αναφερθώ στα μέλη του αντικειμένου.

Όταν τα πράγματα μεγαλώνουν…

- Όπως τα αντικείμενα είναι ομαδοποιήσεις από συναρτήσεις και ιδιότητες, έτσι και διαφορετικά **αρχεία** μπορούν να μας βοηθήσουν να οργανώσουμε τον κώδικά μας.

- Κάθε αρχείο μπορεί να περιέχει τον ορισμό ενός ή περισσότερων αντικειμένων και δεν υπάρχει περιορισμός, ούτε στην ποσότητα του κώδικα που μπορεί να έχει ένα αρχείο, ούτε στο πόσα αρχεία μπορούμε να χρησιμοποιήσουμε σε μία εφαρμογή.



Όταν τα πράγματα μεγαλώνουν... διασπάμε το πρόγραμμα σε περισσότερα αρχεία...

```
var die = {
    size: 6,
    totalRolls: 0,
    roll: function(){
        var result = Math.ceil(this.size * Math.random());
        this.totalRolls += 1;
        return result;
    }
};


die.size = 10;
console.log(die.roll());
console.log(die.roll());
console.log(die.roll());
console.log("Total rolls " + die.totalRolls);

console.log(die);
```

program.js: Κεντρικό

```
die.size = 10;
console.log(die.roll());
console.log(die.roll());
console.log(die.roll());
console.log("Total rolls " + die.totalRolls);

console.log(die);
```

dice.js:

```
var die = {
    size: 6,
    totalRolls: 0,
    roll: function(){
        var result = Math.ceil(this.size * Math.random());
        this.totalRolls += 1;
        return result;
    }
};
```

Και πρέπει με κάποιο τρόπο να ενωθούν! (1/4)

Ψάξε στον ίδιο κατάλογο

Ψάξε για αρχείο με όνομα dice

```
require("./dice");
```

program.js

```
die.size = 10;
console.log(die.roll());
console.log(die.roll());
console.log(die.roll());
console.log("Total rolls " + die.totalRolls);

console.log(die);
```

Το να καλέσω απλά με τη συνάρτηση **require** το **dice.js** δε θα μου δώσει πρόσβαση σε όσα εμπεριέχει.

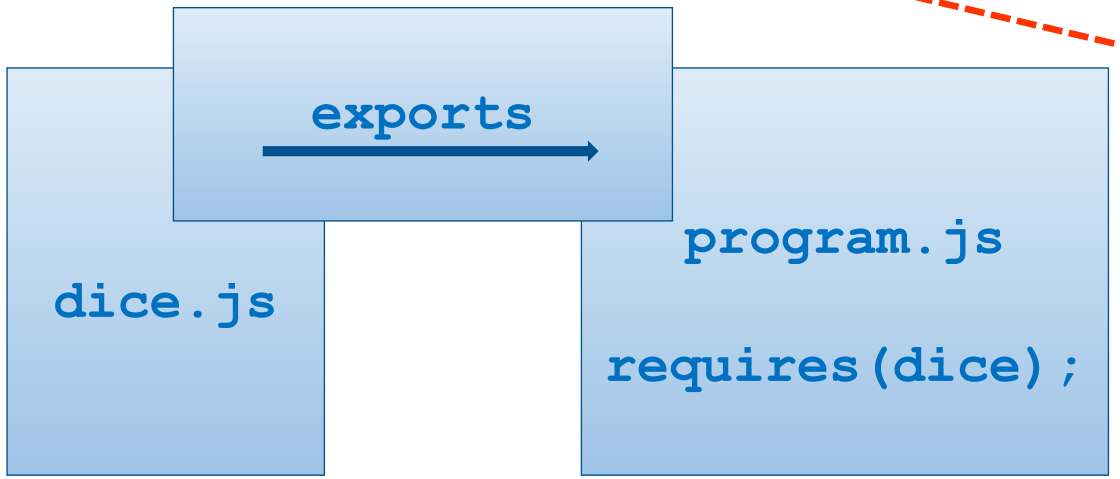
- Για να περιοριστεί η πολυπλοκότητα από προεπιλογή δεν είναι διαθέσιμα όσα είναι μέσα σε ένα αρχείο.
- Αυτό υπάρχει στις περισσότερες γλώσσες και λέγεται **encapsulation** (**ενθυλάκωση**).

```
C:\examples>node program.js
C:\examples\program.js:3
die.size = 10;
^

ReferenceError: die is not defined
    at Object.<anonymous> (C:\examples\program.js:3:1)
    at Module._compile (module.js:570:32)
    at Object.Module._extensions..js (module.js:579:10)
    at Module.load (module.js:487:32)
```

Και πρέπει με κάποιο τρόπο να ενωθούν! (2/4)

dice.js

```
var die = {
    size: 6,
    totalRolls: 0,
    roll: function(){
        var result = Math.ceil(this.size * Math.random());
        this.totalRolls += 1;
        return result;
    }
};

exports.game = die;
```



- Δημιουργούμε ένα νέο property (**game**) για το αντικείμενο **exports** και του αναθέτουμε την τιμή die.
- Οτιδήποτε είναι διαθέσιμο στο **exports** είναι προσβάσιμο από εξωτερικό αρχείο.

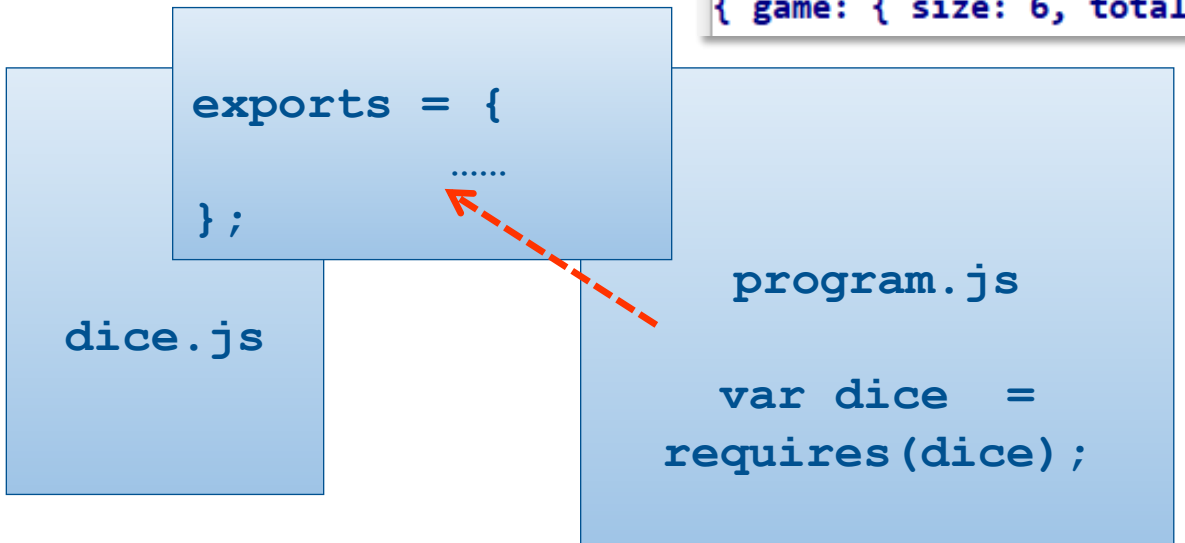
Και πρέπει με κάποιο τρόπο να ενωθούν! (3/4)

program.js

```
var dice = require("./dice");

console.log(dice);

//die.size = 10;
//console.log(die.roll());
//console.log(die.roll());
//console.log(die.roll());
//console.log("Total rolls " + die.totalRolls);

//console.log(die);
```

- Για να αποκτήσω πρόσβαση στο αντικείμενο **exports** θα πρέπει να ορίσω μία μεταβλητή της οποίας θα της αναθέσω την τιμή που επιστρέφει η συνάρτηση require.

```
C:\examples>node program.js
{ game: { size: 6, totalRolls: 0, roll: [Function: roll] },
```



Και πρέπει με κάποιο τρόπο να ενωθούν! (4/4)

program.js

```
var dice = require("./dice");
var die = dice.game;

die.size = 10;
console.log(die.roll());
console.log(die.roll());
console.log(die.roll());
console.log("Total rolls " + die.totalRolls);

console.log(die);
```

- Ορίζοντας και τη μεταβλητή **die** έχω το αποτέλεσμα που θέλω.

dice.js

```
var die = {
    size: 6,
    totalRolls: 0,
    roll: function(){
        var result = Math.ceil(this.size * Math.random());
        this.totalRolls += 1;
        return result;
    }
};
exports.game = die;
exports.name = "My dice exports";
```

Προσέξτε ότι μπορώ να ορίσω και άλλα properties στο export

```
C:\examples>node program.js
5
5
6
Total rolls 3
{ size: 10, totalRolls: 3, roll: [Function: roll] }
```

Τι είδαμε μέχρι τώρα;

- Είδαμε τα αντικείμενα (objects) και δημιουργήσαμε δικά μας αντικείμενα.
- Μάθαμε για τις ιδιότητες των αντικειμένων, που τους επιτρέπουν να έχουν κατάσταση, καθώς και τις συναρτήσεις.
- Είδαμε τη χρήση της δεσμευμένης λέξης this εντός ενός αντικειμένου.
- Τέλος, μάθαμε τη χρήση των require και exports για να γράφουμε κώδικα σε διαφορετικά αρχεία. Με αυτό τον τρόπο μπορούμε να χρησιμοποιήσουμε και έτοιμο κώδικα που έχουν γράψει άλλοι προγραμματιστές. Υπάρχουν χιλιάδες βιβλιοθήκες JavaScript έτοιμες για να τις χρησιμοποιήσετε.

Έλεγχος – Testing

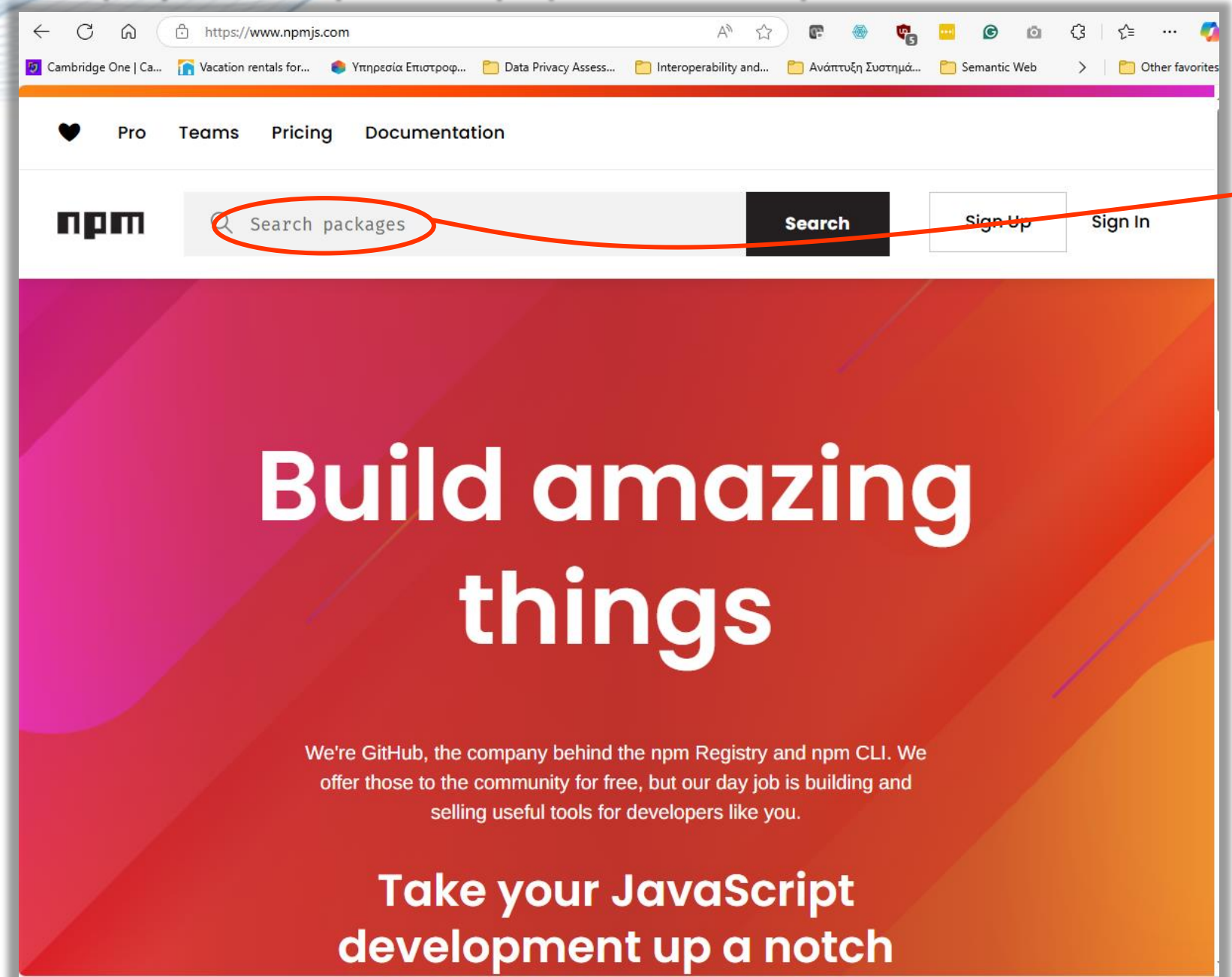
- Όταν γράφουμε κώδικα θα πρέπει να ελέγξουμε αν δουλεύει σωστά.
- Bugs (σφάλματα) υπάρχουν στα περισσότερα προγράμματα.
- Η διαδικασία ελέγχου ενός προγράμματος για το αν δουλεύει σωστά κάτω από όλες τις συνθήκες ονομάζεται testing.
- Υπάρχουν διάφορες τεχνικές για testing.
  - Για απλά προγράμματα μπορούμε να τα ελέγξουμε με το να τα τρέξουμε απλά και να δούμε τα αποτελέσματα που βγάζουν.
  - Αυτό που θα δούμε στη συνέχεια είναι ο **αυτόματος έλεγχος** (automated testing). Σε αυτή την περίπτωση ο έλεγχος γίνεται από ένα άλλο πρόγραμμα για να καλυφθούν όλες οι περιπτώσεις.

Ας φτιάξουμε ένα βαθμολόγιο… και πρώτα απ’ όλα οργάνωση!

```
C:\examples>mkdir gradebook

C:\examples>cd gradebook

C:\examples\gradebook>dir
 Volume in drive C has no label.
 Volume Serial Number is 0050-C176

 Directory of C:\examples\gradebook

03/11/2017  12:16 μμ    <DIR>          .
03/11/2017  12:16 μμ    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)  28.506.210.304 bytes free

C:\examples\gradebook>mkdir tests

C:\examples\gradebook>mkdir lib

C:\examples\gradebook>dir
 Volume in drive C has no label.
 Volume Serial Number is 0050-C176

 Directory of C:\examples\gradebook

03/11/2017  12:17 μμ    <DIR>          .
03/11/2017  12:17 μμ    <DIR>          ..
03/11/2017  12:17 μμ    <DIR>          lib
03/11/2017  12:17 μμ    <DIR>          tests
               0 File(s)              0 bytes
               4 Dir(s)  28.506.210.304 bytes free
```

- Οργάνωση κώδικα
  - **`/gradebook`**: ο κώδικας του προγράμματός μου
  - **`/gradebook/lib`**: οι βιβλιοθήκες της εφαρμογής μου (επαναχρησιμοποιήσιμος κώδικας).
  - **`/gradebook/tests`**: ο κώδικας για τους ελέγχους της εφαρμογής.

Λίγη ακόμα οργάνωση… Node Package Manager

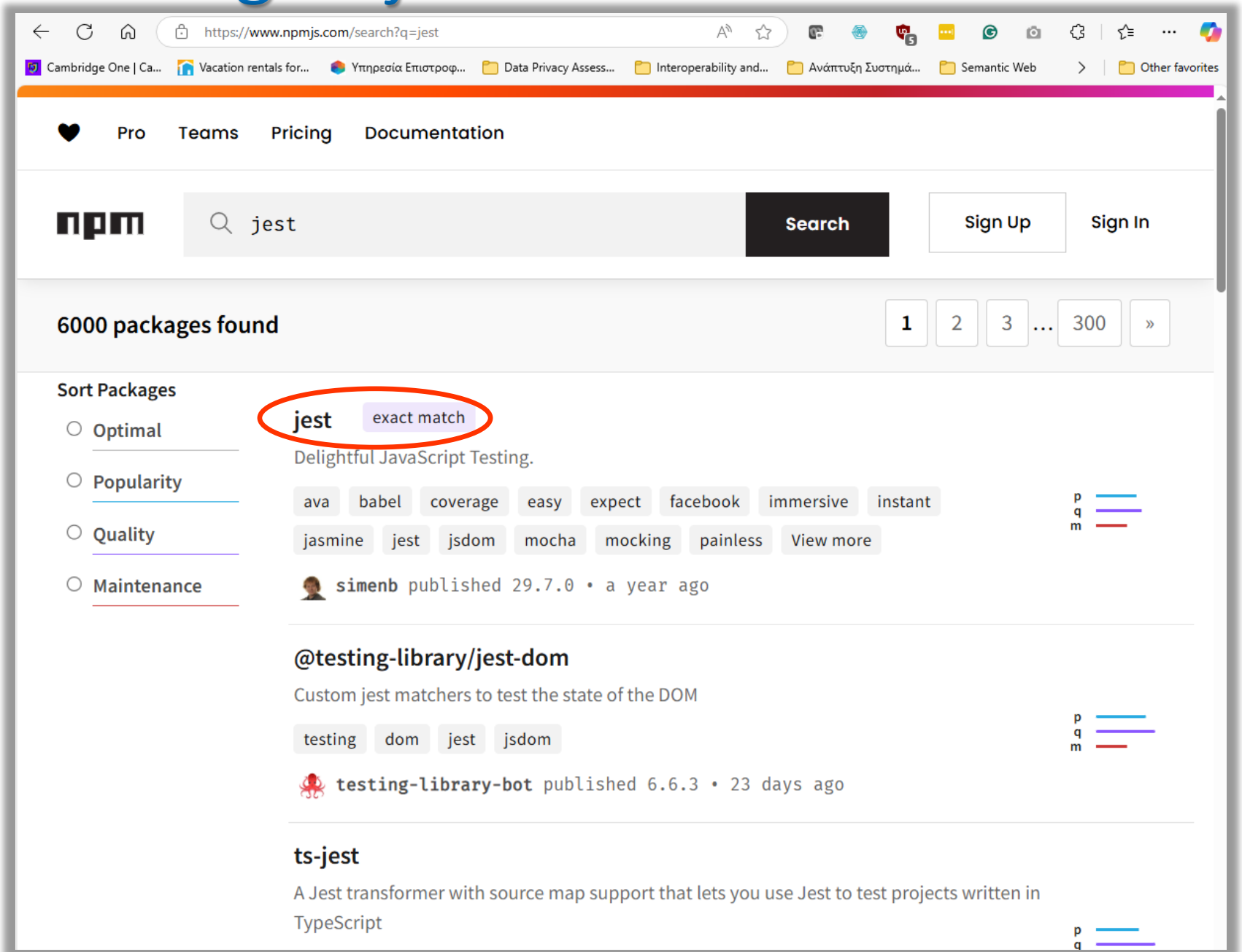


Υπάρχουν modules του node για:

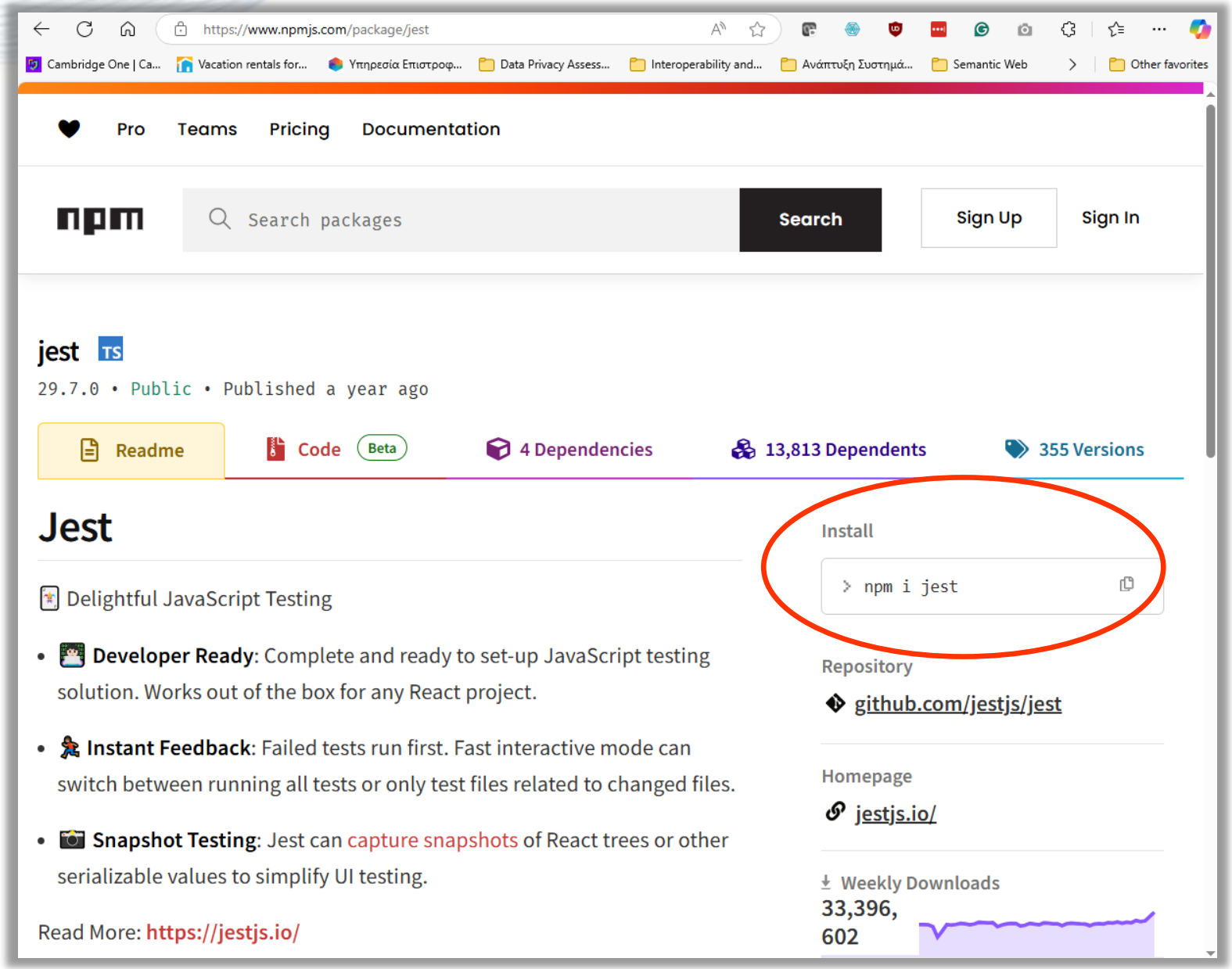
- Δημιουργία Web Site,
- Αποθήκευση δεδομένων,
- Εκτέλεση αυτόματων ελέγχων,
- Κρυπτογράφηση δεδομένων και πολλά, πολλά ακόμα…

Node Package Manager: jest

- Το Jest είναι ένα πρόγραμμα για έλεγχο μονάδων προγράμματος – προγράμματα ή βιβλιοθήκες (unit testing).



jest: Εγκατάσταση



Στο command prompt εκτελέστε:

```
npm i jest
```

jest στην πράξη…

package.json grades.js

```
{
  "name": "gradebook",
  "version": "1.0.0",
  "description": "Example project for Jest testing",
  "main": "index.js",
  "scripts": {
    "test": "jest"
  },
  "author": "Your Name",
  "license": "ISC",
  "devDependencies": {
    "jest": "^29.0.0"
  }
}
```

- Φτιάξτε ένα αρχείο με όνομα **`package.json`** και βάλτε το παραπάνω περιεχόμενο.
- Και είμαστε έτοιμοι να γράψουμε τον κώδικα και τα αντίστοιχα test.

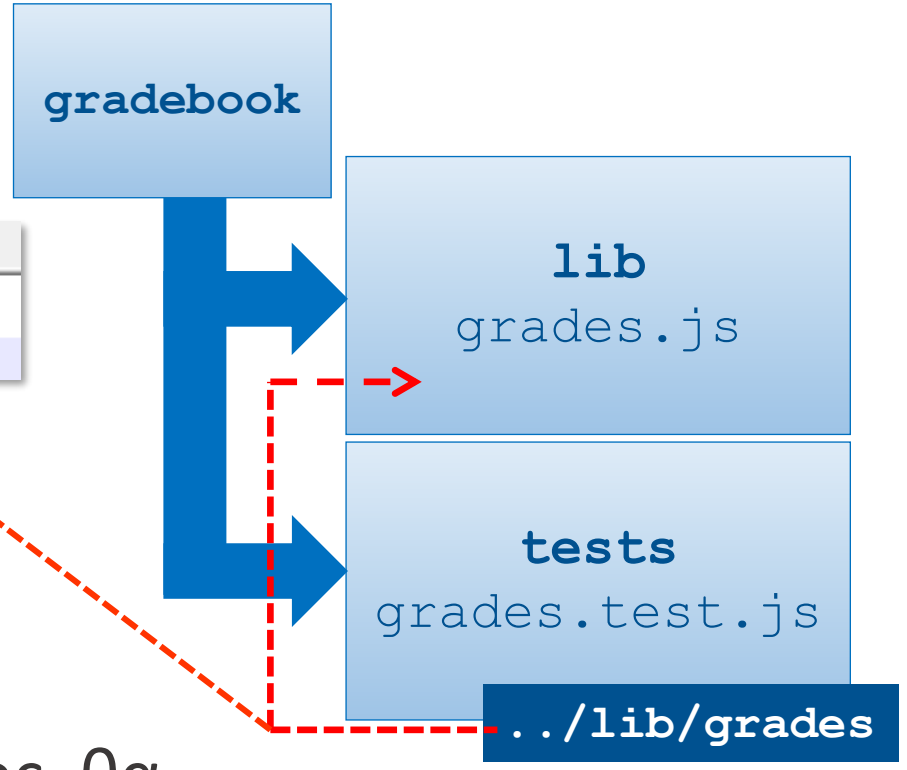
Το πρώτο test

grades.js | package.json

```
var gradeBook = {

};

exports.book = gradeBook;
```

grades.test.js | package.json | grades.js

```
const book = require("../lib/grades").book;
```



- Φτιάχνω τα δύο πρώτα μου αρχεία:
  - Το **grades.js**, στον κατάλογο **lib**
  - Το **grades.test.js** στον κατάλογο **tests**

- Και πριν γράψω τον κώδικα του προγράμματος, θα ξεκινήσω πρώτα σχεδιάζοντας τον έλεγχο.
  - Θα μας βοηθήσει αυτό να σχεδιάσουμε σωστότερα το πρόγραμμα.

Ας ξεκινήσουμε να το γράφουμε...

grades.test.js | package.json | grades.js

```
const book = require("../lib/grades").book;

test("Can add new grade", () => {

});
```

Οι έλεγχοι είναι συναρτήσεις που καλεί το jest.

Η `test` παρέχεται από το Jest.

Έτσι ονομάζουμε το test μας.
Είναι εναλλακτικός τρόπος για να προσθέσουμε κάτι στο αντικείμενο (προσέξτε ότι μπορούμε να βάλουμε κενά).

```
C:\Users\tzimasPC\Desktop\Programming Techniques\Lecture 05\gradebook>npm test

> gradebook@1.0.0 test
> jest

 PASS  tests/grades.test.js
  √ Can add new grade (1 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.621 s, estimated 3 s
Ran all test suites.
```

... το τεστ μας.

grades.test.js | package.json | grades.js

```
const book = require("../lib/grades").book;

test("Can add new grade", () => {
    book.addGrade(90);
    const count = book.getCountOfGrades();
    expect(count).toBe(1);
});
```

Assertion (**ισχυρισμός**):
Αυτό είναι ένα assertion για το Jest.

Απέτυχε στη γραμμή 4.
Γιατί;

```
C:\Users\tzimasPC\Desktop\Programming Techniques\Lecture 05\gradebook>npm test

> gradebook@1.0.0 test
> jest

 FAIL  tests/grades.test.js
  × Can add new grade (2 ms)

  ● Can add new grade

    TypeError: book.addGrade is not a function

      2 |
      3 | test("Can add new grade", () => {
    > 4 |     book.addGrade(90);
        |          ^
      5 |     const count = book.getCountOfGrades();
      6 |     expect(count).toBe(1);
      7 | });

      at Object.addGrade (tests/grades.test.js:4:10)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        3.129 s
Ran all test suites.
```

Το τελικό βήμα...

Ο χαρακτήρας «_» μπροστά από ένα μέλος αντικειμένου είναι μία σύμβαση για όσα μέλη θέλουμε να χρησιμοποιούνται εσωτερικά στο αντικείμενο.

grades.test.js | package.json | grades.js | new 2

```
var gradeBook = {

    _grades: [],

    addGrade: function(newGrade) {
        this._grades.push(newGrade);
    },

    getCountOfGrades: function() {
        return this._grades.length;
    },
};

exports.book = gradeBook;
```

```
C:\Users\tzimasPC\Desktop\Programming Techniques\Lecture 05\gradebook>

> gradebook@1.0.0 test
> jest

 PASS  tests/grades.test.js
  √ Can add new grade (3 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.605 s, estimated 3 s
Ran all test suites.
```

grades.test.js | package.json | grades.js

```
const book = require("../lib/grades").book;

test("Can add new grade", () => {
    book.addGrade(90);
    const count = book.getCountOfGrades();
    expect(count).toBe(1);
});
```

Η push είναι μία συνάρτηση διαθέσιμή για arrays, ώστε να βάζουμε δεδομένα.

Η length είναι μία ιδιότητα διαθέσιμή για arrays.

Και το δεύτερο test… Βήμα 1

- Ας βάλουμε στο βαθμολόγιο περισσότερους βαθμούς και τη δυνατότητα να υπολογίζει μέσους όρους.

grades.test.js | package.json | grades.js | new 2

```
const book = require("../lib/grades").book;

test("Can add new grade", () => {
    book.addGrade(90);
    const count = book.getCountOfGrades();
    expect(count).toBe(1);
});

test("Can average grades", () => {
    book.addGrade(100);
    book.addGrade(50);
    const average = book.getAverage();
    expect(average).toBe(75);
});
```

Και το δεύτερο test... Βήμα 2

grades.test.js | package.json | grades.js | new 2

```
var gradeBook = {

    _grades: [],

    addGrade: function(newGrade) {
        this._grades.push(newGrade);
    },

    getCountOfGrades: function() {
        return this._grades.length;
    },

    getAverage: function() {
        var total = 0;
        for(var i = 0; i < this._grades.length; i += 1) {
            total += this._grades[i];
        }
        return total / this._grades.length;
    },

};

exports.book = gradeBook;
```

Το ίδιο θα μπορούσε να γίνει και με while loop, αλλά τα for loops χρησιμοποιούνται συνήθως αν θέλουμε να διαπεράσουμε ένα array.

Τι πάει στραβά;

grades.test.js | package.json | grades.js | new 2

```
var gradeBook = {

    _grades: [],

    addGrade: function(newGrade) {
        this._grades.push(newGrade);
    },

    getCountOfGrades: function() {
        return this._grades.length;
    },

    getAverage: function() {
        var total = 0;
        for(var i = 0; i < this._grades.length; i += 1) {
            total += this._grades[i];
        }
        return total / this._grades.length;
    },

};

exports.book = gradeBo
```

grades.test.js | package.json | grades.js | new 2

```
const book = require("../lib/grades").book;

test("Can add new grade", () => {
    book.addGrade(90);
    const count = book.getCountOfGrades();
    expect(count).toBe(1);
});

test("Can average grades", () => {
    book.addGrade(100);
    book.addGrade(50);
    const average = book.getAverage();
    expect(average).toBe(75);
});
```

```
FAIL tests/grades.test.js
  √ Can add new grade (3 ms)
  × Can average grades (2 ms)

  ● Can average grades

    expect(received).toBe(expected) // Object.is equality

    Expected: 75
    Received: 80

      11 |     book.addGrade(50);
      12 |     const average = book.getAverage();
    > 13 |     expect(average).toBe(75);
         |                     ^
      14 | });
      15 |

      at Object.toBe (tests/grades.test.js:13:21)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:   0 total
Time:        0.646 s, estimated 1 s
Ran all test suites.
```

Ένα μικρό hint…

grades.test.js | package.json | grades.js | new 2

```
const book = require("../lib/grades").book;

test("Can add new grade", () => {
    book.addGrade(90);
    const count = book.getCountOfGrades();
    expect(count).toBe(1);
});

test("Can average grades", () => {
    book.addGrade(100);
    book.addGrade(50);
    const average = book.getAverage();
    expect(average).toBe(75);
});
```

Η λύση… grades.test.js

grades.test.js | package.json | grades.js | new 2

```
const book = require("../lib/grades").book;

beforeEach(() => {
    // Επαναφορά της κατάστασης πριν από κάθε test
    book.reset();
});

test("Can add new grade", () => {
    book.addGrade(90);
    const count = book.getCountOfGrades();
    expect(count).toBe(1);
});

test("Can average grades", () => {
    book.addGrade(100);
    book.addGrade(50);
    const average = book.getAverage();
    expect(average).toBe(75);
});
```

Χαρακτηριστικό του Jest για την προετοιμασία του περιβάλλοντος ελέγχου.

Αυτή η συνάρτηση θα κληθεί πριν από κάθε τεστ.

Η λύση… grades.js

grades.test.js | package.json | grades.js | new 2

```
var gradeBook = {

    _grades: [],

    addGrade: function(newGrade) {
        this._grades.push(newGrade);
    },

    getCountOfGrades: function() {
        return this._grades.length;
    },

    getAverage: function() {
        var total = 0;
        for(var i = 0; i < this._grades.length; i += 1) {
            total += this._grades[i];
        }
        return total / this._grades.length;
    },

    reset: function() {
        this._grades = [];
    }
};

exports.book = gradeBook;
```

```
C:\Users\tzimasPC\Desktop\Programming Techniques\Lecture 05\gradebook>npm

> gradebook@1.0.0 test
> jest

 PASS  tests/grades.test.js
  √ Can add new grade (3 ms)
  √ Can average grades

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.629 s, estimated 1 s
Ran all test suites.
```

Το κεντρικό πρόγραμμα

Το έχουμε αποθηκεύσει στον κατάλογο gradebook

grades.js | gradeTests.js | program.js

```
1  var book = require("./lib/grades").book;
2
3  for(var i = 2; i < process.argv.length;  i += 1) {
4      book.addGrade(parseInt(process.argv[i]));
5  }
6
7  console.log(book.getAverage());
```

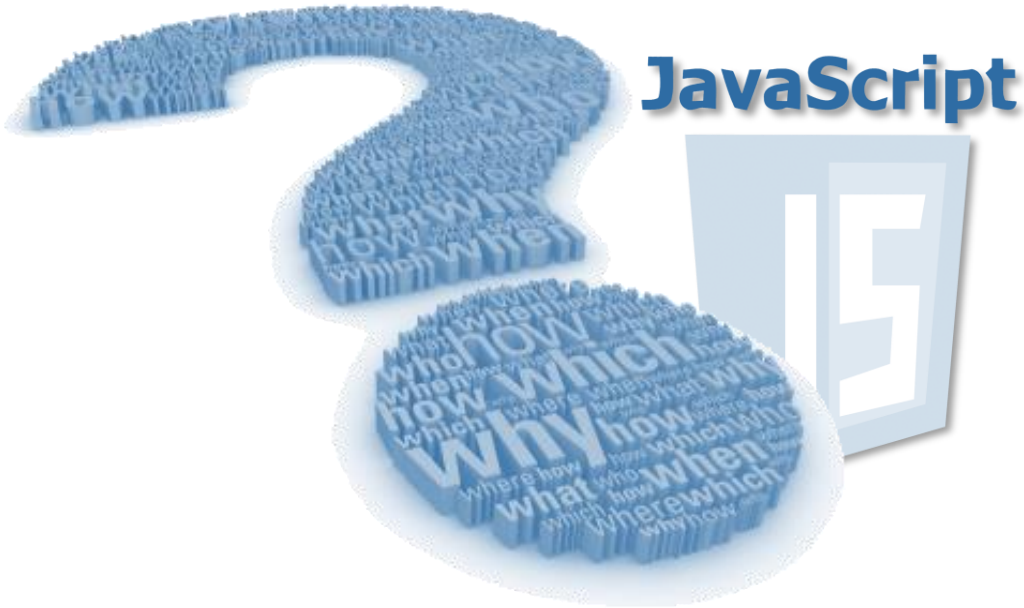
```
C:\examples\gradebook>node program.js 80 91 99 75 43
77.6

C:\examples\gradebook>node program.js 80 88 99 75 43
77

C:\examples\gradebook>node program.js 15 89 74 5 45
45.6
```

Τι είδαμε μέχρι τώρα;

- Μέχρι εδώ χρησιμοποιήσαμε τις γνώσεις που αποκτήσαμε για να φτιάξουμε ένα μικρό βαθμολόγιο και τα σχετικά με αυτό test.

- Η διαδικασία του να γίνουν έλεγχοι σε ένα πρόγραμμα είναι ιδιαίτερα δύσκολή, αλλά κάναμε μία εισαγωγή στο πως μπορεί να γίνει αυτό.

- Το να μάθετε να γράφετε test θα σας κάνει σίγουρα καλύτερους προγραμματιστές!


JavaScript