Ανάπτυξη Κινητών Εφαρμογών

Εργαστήριο

Δημήτρης Ρίγγας

riggas@ionio.gr

Τμήμα Πληροφορικής

Ιόνιο Πανεπιστήμιο



Εργαστήριο 1 – Βασικές αρχές JavaScript

Στόχος: Να εξοικειωθείτε με τη σύγχρονη σύνταξη και λογική της JavaScript (ES6+) — ό,τι θα χρειαστούμε για React Native.



JavaScript - προέλευση

Η `JavaScript` δημιουργήθηκε το 1995, από τη Netscape, ως ένας τρόπος να προστεθεί εκτελέσιμος κώδικας στις ιστοσελίδες και να τις κάνει *ζωντανές*.

Αρχικά ονομαζόταν LiveScript, αλλά λόγω της δημοφιλίας της Java την εποχή μετονομάστηκε σε JavaScript.
Δεν έχει όμως καμία άλλη σχέση με τη Java

Με τη διάδοσή της σε άλλους browsers (*και αφού αποτέλεσε σημαντικό πεδίο μάχης την περίοδο των browser wars*) προτυποποιήθηκε ως `ECMAScript`.

- ✓ Σήμερα οι όροι `JavaScript` και `ECMAScript` χρησιμοποιούνται συχνά ο ένας στη θέση του άλλου.
- ✓ Από το 2015 (**ES6** / ES2015) βγαίνει νέα έκδοση **κάθε χρόνο** — «σύγχρονη JavaScript» = ES6+

JavaScript - χαρακτηριστικά

Γλώσσα προγραμματισμού υψηλού επιπέδου, **δυναμικά τυποποιημένη** (dynamically typed)

Διερμηνευόμενη (interpreted) γλώσσα — στην πράξη οι σύγχρονες engines κάνουν **JIT compilation**

JavaScript δεν τρέχει μόνο μέσα στους browsers.

- ✓ εκτελείται σε όποιο περιβάλλον περιλαμβάνει μια συμβατή `JavaScript engine`, πχ
  - ✓ V8 (Google), αξιοποιείται σε:
    - ✓ Chrome, Opera, Edge, κ.α.
    - ✓ Node.js, Deno, κ.α.
  - ✓ SpiderMonkey (Mozilla)
    - ✓ Firefox
    - ✓ Netscape (παλιότερα)
  - ✓ JavaScriptCore (Apple)
    - ✓ Safari
    - ✓ WebKit browsers
  - ✓ **Hermes** (Meta)
    - ✓ React Native 📱

- ✓ Βασίζεται σε **συμβάντα** και είναι **μονονηματικό**
- ✓ Το σύγχρονο JS χρησιμοποιεί **modules**, **arrow functions** και **promises**

.

(άνω τελεία)

τι θυμόμαστε από τις Τεχνολογίες Διαδικτύου;

(εδώ ας δούμε μερικά ενδιαφέροντα, μόνο)



Μεταβλητές: `const`, `let` (και `var`)

`const` — δεν ξαναγίνεται ανάθεση

```
const name = "Eleni";
name = "Maria";   // ❌ TypeError

const user = { name: "Eleni" };
user.name = "Maria"; // ✅ το αντικείμενο
                     // αλλάζει, η αναφορά όχι
```

`let` — αλλάζει τιμή

```
let count = 0;
count = count + 1;   // ✅
```

`var` — παλιός τρόπος, **αποφεύγεται**
(function scope, όχι block scope)

Κανόνας: `const` παντού· `let` μόνο όταν **πρέπει** να αλλάξει η τιμή· ποτέ `var`.
Ποτέ ανάθεση χωρίς δήλωση (`x = 5`) — σε modules είναι `ReferenceError`.

Τύποι, ισότητα & truthy / falsy

```
typeof 42           // 'number'
typeof "hi"         // 'string'
typeof true         // 'boolean'
typeof undefined    // 'undefined'
typeof null         // 'object' (ιστορικό bug!)
typeof [1, 2]       // 'object'
typeof {}           // 'object'
typeof (() => 1)    // 'function'
Array.isArray([1]) // true
```

Πάντα `===` (χωρίς μετατροπή τύπων)

```
0 == ""       // true  😱
0 === ""      // false ✅
"5" == 5      // true  😱
"5" === 5     // false ✅
```

Falsy: `false`, `0`, `""`, `null`, `undefined`, `NaN`
Όλα τα υπόλοιπα είναι **truthy** (και τα `[]`, `{}`!)

Συναρτήσεις

- Μια συνάρτηση είναι ένα block κώδικα που μπορεί να κληθεί.
  - Δεν δηλώνει επιστρεφόμενο τύπο, ή αν επιστρέφει κάτι
  - Μπορεί να επιστρέφει κάτι, μπορεί και όχι (τότε επιστρέφει `undefined`)
  - Μπορεί να λαμβάνει κάποιες παραμέτρους, προσδιορίζοντας μόνο τα ονόματά τους, όχι τύπους
  - Μπορεί να έχει όνομα, μπορεί όμως να είναι και ανώνυμη
  - Είναι **τιμή**: μπορεί να δοθεί ως παράμετρος σε άλλη συνάρτηση (callback)

```
function hello(){   // Definition
    console.log("Hello");
}
hello();            // Call
```

```
function greetingStr(name){
    return "Hello "+name;
}
console.log( greetingStr("Eleni") );
```

```
const f = function(){
    console.log("This is anonymous");
};
f(); // prints: This is anonymous
const g = f;
g(); // prints: This is anonymous
```

```
const f = function(){
    console.log("This is anonymous");
};  // ⚠ χωρίς ; η επόμενη ( καλεί την f!
(function(fun) {
    console.log("Calling another function");
    fun();
})(f);  // Defining and calling with a param
```

Συναρτήσεις & εμβέλεια

- Η χρήση `let` και `const` για την δήλωση μεταβλητών και σταθερών αυτές είναι local στο block που ορίζονται, αλλά προσοχή..
  - ό,τι ορίζεται σε ένα block είναι local εκεί

```
let x=10;
for(let i = 0; i < 10; i++){
    x += i;
}
console.log(x); // prints 55
typeof i;       // returns 'undefined'
```

  - μεταβλητές με το ίδιο όνομα σε εσωτερικό block κρύβουν άλλες σε πιο εξωτερικά

```
let x = 10;
function multiply(z){
    let x = 5;
    return x * z;
}
multiply(x); // returns 50, not 100
typeof z;    // returns 'undefined'
```

  - μια function αναζητά σε πιο εξωτερικά block

```
let x = 10;
function increase(){
    x +=1 ;
}
increase();
console.log(x); // prints 11
```

  - Οι δηλώσεις `function` «ανεβαίνουν» στην κορυφή (**hoisting**) — οι `const f = () => …` **όχι**

```
isInScope();   // ✅
function isInScope(){
    console.log("Defined below its call ?!");
}
later();       // ❌ ReferenceError
const later = () => console.log("no");
```

Συναρτήσεις arrow

- Οι συναρτήσεις μπορούν να οριστούν και με πιο σύντομη σύνταξη:

κλασσική σύνταξη:

```
function hello(){
    console.log("Hello");
}
```

multiline arrow function:

```
const hello = () => {
    console.log("Hello");
};
```

single **expression** arrow function:

```
const hello = () => console.log("Hello");
```

arrow function με παράμετρο:

```
const hello = (name) => console.log("Hello "+name);
```

ή, *για μονή παράμετρο*, ακόμη πιο απλά:

```
const hello = name => console.log("Hello "+name);
```

arrow function επιστρεφόμενη τιμή (χωρίς `return`):

```
const hello = () => "Hello World!";
```

επιστροφή **αντικειμένου**: σε παρενθέσεις!

```
const mk = (id) => ({ id, done: false });
```

Παράμετροι συναρτήσεων

- Αν κατά την κλήση μιας συνάρτησης δοθούν περισσότερες παράμετροι από αυτές που δηλώνει η συνάρτηση, αυτές αγνοούνται χωρίς μήνυμα λάθους.
- Μια συνάρτηση μπορεί να δηλώνει κάποιες παραμέτρους ως προαιρετικές, δίνοντας για καθεμία την τιμή που θα πρέπει να χρησιμοποιείται αν δεν παρέχεται κατά την κλήση
  - ή μπορεί εντός της συνάρτησης να υπάρχει έλεγχος για το εάν δόθηκε κάποια παράμετρος και αναλόγως να εκτελείται ο αντίστοιχος κώδικας

```
const power = (num, exp=2) => num ** exp ;
power(3);     // returns 9
power(2, 10) // returns 1024
```

```
function minus(a, b) {
  if (b === undefined) return -a;
  return a - b;
}
minus(10);     // returns -10
minus(10, 5);  // returns  5
```

Πρότυπες συμβολοσειρές (Template Strings ή Literals)

Οριοθετούνται με χαρακτήρες backtick `` ` ``, επιτρέποντας συμβολοσειρές πολλαπλών γραμμών, παρεμβολή συμβολοσειρών με ενσωματωμένες εκφράσεις και ειδικές κατασκευές που ονομάζονται πρότυπα με ετικέτες.
Μερικά παραδείγματα:

```
`string text`

`string text line 1
 string text line 2`

`string text ${expression} string text`

tagFunction`string text ${expression} string text`
```

Πρότυπες συμβολοσειρές (Template Strings ή Literals)

Εκτύπωση συμβολοσειράς πολλαπλών γραμμών

```
console.log(`string text line 1
string text line 2`);
```

```
string text line 1
string text line 2
```

Ενσωμάτωση εκφράσεων (χωρίς concatenation)

```
const a = 5;
const b = 10;
console.log(`Fifteen is ${a + b} and not ${2 * a + b}.`);
```

```
Fifteen is 15 and not 20.
```

Πρότυπα με ετικέτες

```
const person = "Mike";
const age = 28;
function myTag(strings, personExp, ageExp) {
  const str0 = strings[0]; // "That "
  const str1 = strings[1]; // " is a "
  const str2 = strings[2]; // "."
  const ageStr = ageExp < 100 ? "youngster" : "centenarian";
  // We can even return a string built using a template literal
  return `${str0}${personExp}${str1}${ageStr}${str2}`;
}
const output = myTag`That ${person} is a ${age}.`;
console.log(output);
```

Το πρώτο όρισμα της συνάρτησης ετικέτας περιέχει μια σειρά από τιμές συμβολοσειρών.
Τα υπόλοιπα ορίσματα σχετίζονται με τις εκφράσεις/ παραμέτρους.

```
That Mike is a youngster.
```

Αποδόμηση (Destructuring)

Η σύνταξη αποδόμησης καθιστά δυνατή την αποσυσκευασία τιμών από πίνακες ή ιδιοτήτων από αντικείμενα σε ξεχωριστές μεταβλητές.

Αντικείμενα (με το όνομα της ιδιότητας)

```
const user = { id: 7, username: "mark" };
const { id, username } = user;
console.log(`${username}'s ID is ${id}`)
// mark's ID is 7

// μετονομασία & default τιμή
const { username: nick, role = "student" } = user;
```

Στις παραμέτρους συναρτήσεων

```
function Card({ title, year }) { /* ... */ }
Card({ title: "Dune", year: 1965 });
```

Πίνακες (με τη θέση)

```
const colors = ["red", "green", "blue"];
const [first, second] = colors;
// first = "red", second = "green"

const [, , third] = colors;   // "blue"
```

📱 Θα το δείτε **συνέχεια** στο React:

```
const [count, setCount] = useState(0);
```

Spread & Rest: `...`

Spread — «άπλωσε» τα στοιχεία

```
const a = [1, 2];
const b = [...a, 3];          // [1, 2, 3]
const c = [0, ...a];          // [0, 1, 2]

const user = { name: "Eleni", age: 20 };
const older = { ...user, age: 21 };
// { name: "Eleni", age: 21 }
// το user ΔΕΝ άλλαξε
```

Rest — «μάζεψε» τα υπόλοιπα

```
const sum = (...nums) =>
  nums.reduce((acc, n) => acc + n, 0);
sum(1, 2, 3);                    // 6

const [head, ...tail] = [1, 2, 3];
// head = 1, tail = [2, 3]

const { age, ...rest } = user;
// rest = { name: "Eleni" }
```

📱 Έτσι ενημερώνουμε state στο React: **νέο** αντίγραφο, όχι αλλαγή του παλιού

Πίνακες: `map`, `filter`, `find`, `reduce`

Επιστρέφουν **νέα** τιμή — δεν αλλάζουν τον αρχικό πίνακα

```
const courses = [
  { id: 1, title: "Mobile Apps", ects: 6, done: false },
  { id: 2, title: "Web Tech",    ects: 5, done: true  },
  { id: 3, title: "Databases",   ects: 6, done: true  },
];

courses.map(c => c.title);                // ["Mobile Apps", "Web Tech", "Databases"]
courses.filter(c => c.done);              // τα 2 ολοκληρωμένα
courses.find(c => c.id === 2);            // { id: 2, title: "Web Tech", ... }
courses.reduce((sum, c) => sum + c.ects, 0);   // 17
courses.some(c => !c.done);               // true

// Αλυσίδα: τίτλοι ολοκληρωμένων, αλφαβητικά
courses.filter(c => c.done).map(c => c.title).toSorted();
```

⚠️ `push`, `pop`, `splice`, `sort` **αλλάζουν** τον πίνακα · `toSorted()` επιστρέφει νέο (ES2023)
📱 Το `.map()` είναι ο τρόπος που κάνουμε render λίστες στο React

Τιμές vs Αναφορές (references)

Primitives αντιγράφονται

```
let a = 5;
let b = a;
b = 10;
console.log(a);   // 5
```

Αντικείμενα / πίνακες: αντιγράφεται η **αναφορά**

```
const list = [1, 2];
const same = list;
same.push(3);
console.log(list);   // [1, 2, 3] 😱
```

Σύγκριση = «είναι το **ίδιο** αντικείμενο;»

```
[1, 2] === [1, 2]        // false
const x = [1, 2];
x === x                  // true
```

Αντίγραφο (shallow) με spread

```
const copy = [...list];
copy.push(4);            // το list δεν αλλάζει
```

📘 Το React καταλαβαίνει ότι άλλαξε το state **μόνο** αν πάρει **νέα** αναφορά

Αντικείμενα: σύγχρονη σύνταξη

Shorthand properties

```
const name = "Eleni", age = 20;
const user = { name, age };
// αντί για { name: name, age: age }
```

Optional chaining `?.`

```
const u = { address: null };
u.address.city;     // ❌ TypeError
u.address?.city;    // undefined ✅
```

Ternary & && (σύντομη αξιολόγηση)

```
const label = age >= 18 ? "adult" : "minor";
isAdmin && console.log("Welcome admin");
```

Nullish coalescing `??`

```
const n = 0;
n || 10;   // 10  (το 0 είναι falsy)
n ?? 10;   // 0   (μόνο για null/undefined)
```

📱 Χρήσιμα με δεδομένα από APIs, όπου κάποια πεδία μπορεί να λείπουν

Ασύγχρονο JavaScript

Το JavaScript χρησιμοποιεί ένα **event loop**.
Ασύγχρονες λειτουργίες είναι οι **callbacks**, **promises** και **async/await**.

Το JavaScript εκτελείται σε ένα μόνο νήμα (single thread), που σημαίνει ότι μπορεί να κάνει μόνο ένα πράγμα κάθε φορά. Ωστόσο, καταφέρνει να χειρίζεται πολλαπλές εργασίες ταυτόχρονα, χάρη στον βρόχο συμβάντων.

- Στοίβα κλήσεων
  Παρακολουθεί ποια συνάρτηση εκτελείται εκείνη τη στιγμή.
- Web APIs / Εργασίες παρασκηνίου
  Συναρτήσεις όπως setTimeout, fetch ή event listeners μεταβιβάζονται στο περιβάλλον εκτέλεσης (browser ή Node.js) και η επόμενη εντολή συνεχίζει αμέσως.
- Ουρές (Microtask queue για Promises, Task / Callback queue για timers & events)
  Μόλις ολοκληρωθούν αυτές οι εργασίες παρασκηνίου, οι επιστροφές κλήσης τους τοποθετούνται σε μια ουρά.
- Βρόχος συμβάντων
  Ο βρόχος συμβάντων ελέγχει συνεχώς *Είναι κενή η στοίβα κλήσεων; Εάν ναι, παίρνει την πρώτη επιστροφή κλήσης από την ουρά και την ωθεί στη στοίβα για να εκτελεστεί.*

Ασύγχρονο JavaScript - callbacks

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

setTimeout(() => {
  console.log("Timeout");
}, 0);

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

```
Start
End
Timeout
```

Ακόμα κι αν το setTimeout έχει οριστεί σε 0, εξακολουθεί να περνάει από τον βρόχο συμβάντων και εκτελείται μετά τον σύγχρονο κώδικα.

Ασύγχρονο JavaScript - microtasks vs tasks

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

setTimeout(() => console.log("timeout"), 0);

Promise.resolve()
  .then(() => console.log("promise"));

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

```
A
B
promise
timeout
```

- Πρώτα τρέχει **όλος** ο σύγχρονος κώδικας
- Μετά **όλα** τα microtasks (Promises, `await`)
- Μετά το επόμενο task (timers, events)

Ασύγχρονο JavaScript - promises

```
fetch("https://jsonplaceholder.typicode.com/users")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

```
[
 {
   id: 1,
   name: 'Leanne Graham',
   username: 'Bret',
   email: 'Sincere@april.biz',
   address: {
     street: 'Kulas Light',
     suite: 'Apt. 556',
     city: 'Gwenborough',
     zipcode: '92998-3874',
     geo: [Object]
   },
 }
 ...
]
```

- ✓ Το fetch() επιστρέφει μια υπόσχεση (Promise) που επιλύεται σε ένα αντικείμενο απόκρισης (Response) μόλις ολοκληρωθεί το αίτημα.
- ✓ Το .then() χειρίζεται το αντικείμενο Response που επιλύθηκε από την κλήση fetch().
- ✓ Το res.json() είναι μια μέθοδος που διαβάζει το ρεύμα απόκρισης και το αναλύει ως JSON. Επιστρέφει επίσης ένα Promise, το οποίο επιλύεται στα πραγματικά δεδομένα (σε αυτήν την περίπτωση, μια συστοιχία αντικειμένων χρήστη).
- ✓ Λαμβάνει τα αναλυμένα δεδομένα JSON και το console.log(data) εκτυπώνει τα δεδομένα στην κονσόλα.

Ασύγχρονο JavaScript - async/await

Τα async/await είναι συντακτικά στοιχεία που χρησιμοποιούνται όπου υπάρχουν Promises και κάνουν τον ασύγχρονο κώδικα πιο εύκολο να διαβαστεί και να γραφεί, σαν να ήταν σύγχρονος κώδικας.
Ίδια λειτουργία με πριν:

```
async function getUsers() {
  try {
    const res = await fetch("https://jsonplaceholder.typicode.com/users");
    if (!res.ok) throw new Error(`HTTP ${res.status}`);
    const users = await res.json();
    console.log(users);
  } catch (err) {
    console.error(err);
  }
}
getUsers();
```

πιο κατανοητό συντακτικό

- `await` μόνο μέσα σε `async` συνάρτηση (ή στο top level ενός ES module)
- Μια `async` συνάρτηση **πάντα** επιστρέφει Promise
- Το `fetch` **δεν** αποτυγχάνει σε 404/500 — ελέγχουμε το `res.ok`

Closures

Μια συνάρτηση **θυμάται** τις μεταβλητές του scope όπου δημιουργήθηκε, ακόμη κι αφού αυτό τελειώσει

```
function makeCounter() {
  let count = 0;              // "ιδιωτική"
  return () => {
    count += 1;
    return count;
  };
}

const next = makeCounter();
next();   // 1
next();   // 2
const other = makeCounter();
other();  // 1  (δικό του count)
```

Γιατί μας ενδιαφέρει;

- Κάθε callback (`onPress`, `setTimeout`, `.then`) είναι closure
- «Βλέπει» τις τιμές **τη στιγμή που δημιουργήθηκε**

```
let msg = "old";
const show = () => console.log(msg);
msg = "new";
show();   // "new" — βλέπει τη μεταβλητή,
          // όχι αντίγραφο της τιμής
```

Χρήση modules

- Τα modules σας επιτρέπουν να χωρίσουμε τον κώδικα σε επαναχρησιμοποιήσιμα αρχεία χρησιμοποιώντας export και import.
- Τα modules βοηθούν στην οργάνωση και την ενθυλάκωση της λογικής — επιτρέποντας καθαρότερα, συντηρήσιμα και δοκιμασμένα τμήματα κώδικα.

math.mjs

```
export const add = (a,b) => a+b;              // named
export default function mul(a,b){ return a*b; } // default
```

main.mjs

```
import mul, { add } from "./math.mjs";
console.log(add(2,3), mul(2,3));   // 5 6
```

Εκτέλεση με Node.js:

```
node main.mjs
```

- `.mjs` → ES module
- ή `"type": "module"` στο `package.json` και αρχεία `.js`
- Ένα `default` ανά αρχείο, όσα `named` θέλουμε
- 📱 `export default function App()` στο React Native

Από JavaScript σε React Native

Όσα είδαμε σήμερα, όπως θα τα συναντήσετε στα επόμενα εργαστήρια:

JavaScript	React Native
Array destructuring	`const [count, setCount] = useState(0);`
Object destructuring σε παραμέτρους	`function Card({ title, year }) { ... }` (props)
Arrow functions & callbacks	`onPress={() => setCount(c => c + 1)}`
Spread / immutability	`setItems(prev => [...prev, newItem])`
`.map()`	`{items.map(i => <Text key={i.id}>{i.title}</Text>)}`
Ternary / `&&`	`{isLoading ? <Spinner /> : <List />}`
Template strings	`<Text>{`Γεια σου, ${name}!`}</Text>`
`import` / `export default`	`import { View } from 'react-native';`
async / await	Φόρτωση δεδομένων από APIs

Εγκατάσταση & προετοιμασία

1. Εγκαταστήστε το **Node.js** LTS (**v22.13 +**, προτείνεται v24) → https://nodejs.org/en/download
   - *Το ίδιο θα χρησιμοποιήσουμε για Expo/React Native από το Εργαστήριο 2*
2. Επαληθεύστε την εγκατάσταση

```
node -v
npm -v
```

3. Δημιουργήστε ένα φάκελο εργασίας

```
mkdir lab1 && cd lab1
code .
```

4. Κάθε άσκηση σε δικό της αρχείο, π.χ. `ex1.js`, και εκτέλεση με `node ex1.js`
   - Για `import` / `export` ή top-level `await`: αρχεία `.mjs`

💡 Άσκηση 1 – Μεταβλητές και τύποι

Γράψτε ένα σενάριο που:

1. Δηλώνει 3 μεταβλητές (`name`, `age`, `student`) — με `const` ή `let`; Γιατί;
2. Τις εκτυπώνει χρησιμοποιώντας πρότυπες συμβολοσειρές
3. Χρησιμοποιεί `typeof` για να δείξει κάθε τύπο
4. Συγκρίνει `age == "20"` και `age === "20"` — εξηγήστε τη διαφορά

💡 Άσκηση 2 – Πίνακες & Βρόχοι

Δημιουργήστε έναν πίνακα με τα 5 αγαπημένα σας τραγούδια.
Χρησιμοποιήστε:

- έναν βρόχο **for…of** για να εκτυπώσετε το καθένα
- `.map()` για να δημιουργήσετε έναν πίνακα με τίτλους σε κεφαλαία JavaScript Reference: .map()
- `.filter()` για να κρατήσετε μόνο τους τίτλους με περισσότερους από 10 χαρακτήρες
- Βεβαιωθείτε ότι ο αρχικός πίνακας **δεν** άλλαξε

💡 Άσκηση 3 – Συναρτήσεις

Γράψτε μια συνάρτηση `grade(score)` που:

- επιστρέφει `«Pass»` αν το σκορ είναι ≥ 50
- επιστρέφει `«Fail»` σε κάθε άλλη περίπτωση
  Δοκιμάστε την με διάφορες τιμές.

Επέκταση: Ξαναγράψτε την ως arrow function μίας γραμμής (με ternary), και εφαρμόστε την σε έναν πίνακα σκορ με `.map()`.

💡 Άσκηση 4 – Αντικείμενα & Αποδόμηση

Ορίστε ένα αντικείμενο `book` με `title`, `author`, `year`.
Χρησιμοποιήστε αποδόμηση για να εκτυπώσετε:

> “📘 *title* από *author* (*year*)”

Επέκταση:

- Γράψτε συνάρτηση `describe({ title, author })` που αποδομεί στις παραμέτρους
- Δημιουργήστε `updated` με νέο `year` χρησιμοποιώντας spread — το `book` να μείνει ίδιο

💡 Άσκηση 5 – Async Fetch

Γράψτε μια συνάρτηση που ανακτά χρήστες από `https://jsonplaceholder.typicode.com/users` και εκτυπώνει μόνο τα ονόματά τους.

- Μία φορά με `.then()` και μία με `async/await`
- Τι γίνεται αν αλλάξετε το URL σε `.../userz`; Χειριστείτε το σφάλμα.

💡 Άσκηση 6 – Immutable ενημερώσεις λίστας

Ξεκινήστε από:

```
const todos = [
  { id: 1, text: "Install Node", done: true },
  { id: 2, text: "Learn JS",     done: false },
];
```

Γράψτε συναρτήσεις που **επιστρέφουν νέο πίνακα** χωρίς να αλλάζουν τον αρχικό:

1. `addTodo(todos, text)` — προσθέτει νέο todo (μοναδικό `id`, `done: false`)
2. `toggleTodo(todos, id)` — αντιστρέφει το `done` του todo με αυτό το `id`
3. `removeTodo(todos, id)` — αφαιρεί το todo

Στο τέλος εκτυπώστε `todos` — πρέπει να είναι **αμετάβλητο**.
Hint: `[...arr, x]`, `.map(t => t.id === id ? { ...t, done: !t.done } : t)`, `.filter()`

🚀 Mini-project: Μορφοποίηση δεδομένων JSON

Εργασία:
Δημιουργήστε ένα σενάριο `formatUsers.mjs` που:

1. Ανακτά χρήστες από το `https://jsonplaceholder.typicode.com/users` (με `async/await`)
2. Εξάγει το όνομα, το email και την πόλη (με αποδόμηση)
3. Τα ταξινομεί αλφαβητικά κατά όνομα
4. Τα εκτυπώνει όμορφα μορφοποιημένα:

```
👤 Leanne Graham
📧 Sincere@april.biz
🏙️  Gwenborough
-----------------
```

Bonus: Βάλτε τη μορφοποίηση σε ξεχωριστό module (`format.mjs`) και κάντε `import`.

Πηγές

- Marijn Haverbeke, Eloquent JavaScript, 4th edition (2024), CC BY-NC
  https://eloquentjavascript.net
- JavaScript Guide από Mozilla MDN
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
- JavaScript.info
  https://javascript.info
- React — JavaScript που χρειάζεστε για React
  https://react.dev/learn/javascript-in-jsx-with-curly-braces
- JSFiddle test bed
  https://jsfiddle.net