ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Objects, Functions and Modules

Αθανάσιος Ανδρούτσος

Objects (1)

- Το object είναι μία δομή που παρέχει ένα σύνολο από ιδιότητες (key-value pairs). Τεχνικά είναι ένα instance του τύπου Object, που είναι το root type όλων των τύπων της ιεραρχίας τύπων της JavaScript
- Για παράδειγμα, ο ορισμός ενός object customer θα μπορούσε να γίνει όπως παρακάτω:

```
const customer = {
        name: "J. Papadopoulos",
        age: 32,
        height: 1.90
};
```

- Το *customer* είναι ένα **object.**
- Περιέχει **τρία πεδία,** *name*, *age* και *height*
- Κάθε πεδίο έχει μία τιμή μετά τον τελεστή **:**

Objects (2)

- Τα objects στην JavaScript είναι στην πραγματικότητα instances που **περιέχουν properties** σε μορφή *property*: *value*
- Το **property name** είναι ένα η ιδιότητα (property) και το **value** μία συγκεκριμένη τιμή του συνόλου τύπων δεδομένων της JS

Objects (3)

- Τα properties είναι συνήθως strings (αλλά μπορεί να είναι και Symbol)
- Τα properties ενός object είναι δυναμικά. Μπορούμε να εισάγουμε νέα properties @runtime ή και να διαγράψουμε
- Επίσης κάθε object εκτός από τα properties που το ίδιο ορίζει (**own properties**), κληρονομεί και properties (**inherited**) από το parent prototype object

Κληρονομικότητα (1)

- Στην JavaScript η κληρονομικότητα δουλεύει διαφορετικά από ότι στις 'κλασικές' γλώσσες προγραμματισμού, όπως Java, C++, C#, κλπ.
- Στην JavaScript η κληρονομικότητα είναι prototype-based, όπου **prototype** είναι το όνομα ενός object που είναι μέρος του Constructor function στον οποίο βασίζεται το object
- Το ποιο είναι αυτό το prototypal object αναφέρεται στην ιδιότητα __proto__ (not ES standard) ή **[[prototype]]** του object (not accessible)
- Μπορούμε να πάρουμε την ιδιότητα αυτή με Object.getPrototypeOf(object)

Κληρονομικότητα (2)

- Η JavaScript παρέχει ως μέρος της βιβλιοθήκης της το **Object** constructor function, το οποίο περιέχει μία ιδιότητα, **Object.prototype** που είναι ένα object που περιέχει διάφορες ιδιότητες και μεθόδους
- Κάθε object που ορίζουμε εμείς αυτόματα κληρονομεί από το Object.prototype όλες τις ιδιότητες και μεθόδους (π.χ. Object.prototype.toString())
- Αν επομένως δεν κάνει override την toString() σαν own property, τότε μία κλήση στην toString() θα οδηγήσει στην toString() του Object, που δεν κάνει τίποτα παρά επιστρέφει [object Object]

Object state as string


```
1  const user = {
2      id: 1,
3      firstname: "Alice",
4      lastname: "W."
5  }
6
7  console.log(JSON.stringify(user))
8
9  for (let prop in user) {
10     console.log(`${prop} ${user[prop]}`)
11 }
12
13 for (let [key, value] of Object.entries(user)) {
14     console.log(`${key} - ${value}`)
15 }
```

- Συνήθως δεν χρησιμοποιούμε την toString() για να εκτυπώσουμε το state ενός object
- Μπορούμε να το κάνουμε με **JSON.stringify()** ή με for … in ή με for … of

Δημιουργία objects

- Με **object literal**, δηλαδή comma separated name: value pairs

```
1   const emptyObj = {}
2   const point = {x: 1, y: 2}
3   const point2 = {x: point.x, y: point.y}
```

- Ως value μπορούμε να έχουμε οποιοδήποτε expression

Δημιουργία με new


```
5   const point3D = new Object();
6   point3D.x = 1
7   point3D.y = 2
8   point3D.z = 3
```

- Η JavaScript μας παρέχει constructors για τους build-in types της, όπως new Object(), new Date(), new Map(), new Array()
- Επίσης, όπως βλέπουμε μπορούμε να δημιουργούμε δυναμικά ιδιότητες (properties)

Object.create

- Η Object.create() παίρνει ως 1η παράμετρο το prototype με βάση το οποίο θα δημιουργήσει ένα instance
- Για παράδειγμα, η παρακάτω δήλωση

```
const obj = Object.create(Object.prototype)
```

- Δημιουργεί ένα obj με βάση το Object.prototype που είναι παρόμοιο με το const obj = {}

Πρόσβαση στις ιδιότητες objects

Έστω το object customer:

```
let customer = {
        name: "J. Papadopoulos",
        age: 32,
        height: 1.90
};
```

- Πρόσβαση στις ιδιότητες αντικειμένων (objects)
  - Με τον τελεστή τελεία **.**
    π.χ. customer.name = "A. Androutsos" , ή
  - Με τον τελεστή **[]**
    π.χ. customer["name"] = "A. Androutsos"

Τελεστής [] (1)

- Ο indexer **[]** είναι πιο ευέλικτος από την τελεία **.**
- Ο τελεστής [] μπορεί να λάβει ως παράμετρο **μεταβλητή** όπως στην παρακάτω μορφή της for
- Με την ειδική μορφή της for, την **for .. in** μπορούμε να προσπελάσουμε τις ιδιότητες ενός object:

```
1   const point = {x: 1, y: 2}
2
3   for (let prop in point) {
4       console.log(`${prop}: ${point[prop]}`)
5   }
6
```

Τελεστής [] (2)


```
8   const stocks = { Apple: "AAPL", Microsft: "MSFT", Google: "GOOG" }
9   const portfolio = {AAPL: 10, MSFT: 0, GOOG: 2}
10
11  function addStock(portfolio, stock, shares) {
12      portfolio[stock] += shares;
13  }
14
15
16  addStock(portfolio, stocks.Microsft, 30)
17  console.log(portfolio)
```

- Με τον τελεστή . αναφερόμαστε στα πεδία του object, literally (stocks.Microsoft)
- Με τον τελεστή [] μπορούμε και αναφερόμαστε και μέσω μεταβλητών (portfolio[stock]

Conditional Check

- Τα property access expressions αποτυγχάνουν αν το property είναι undefined

```
31  const book = {author: {firstname: "Th.",  surname: "Androutsos"}}
32  let surname
33
34  // Verbose null/undefined check
35  if (book) {
36      if (book.author) {
37          surname = book.author.surname
38      }
39  }
40
41  // Conditional property access
42  surname = book?.author?.surname
```

Add / Delete Properties


```
31  const book = {author: {firstname: "Th.",  surname: "Androutsos"}}
32  book.author.title = "Dr."
33  delete book.author.firstname
34  console.log(book)
```

- Μπορούμε να προσθέσουμε και να διαγράψουμε δυναμικά properties με add / delete
- Με delete δεν μπορούμε να διαγράψουμε properties που έχουν configurable: false

Own Properties


```
31  const book = {author: {firstname: "Th.",  surname: "Androutsos"}}
32
33  if (book.hasOwnProperty('author')) {
34      console.log(book.author)
35  }
```

- Η *hasOwnProperty*() ελέγχει αν ένα object έχει μία ιδιότητα που δεν είναι inherited

Διάσχιση object (1)


```
31  const book = {author: {firstname: "Th.",  surname: "Androutsos"}}
32
33  for (let prop in book.author) {
34      console.log(`${prop}: ${book.author[prop]}`)
35
36  }
```

```
31  const book = {author: {firstname: "Th.",  surname: "Androutsos"}}
32
33  for (let [k, v] of Object.entries(book.author)) {
34      console.log(`${k}: ${v}`)
35
36  }
```

- Με for/in
- Με Object.keys ή object.entries

Αντιγραφή Objects (Shallow)


```
31  const book = {author: {firstname: "Th.",  surname: "Androutsos"}}
32  const clonedBook = {};
33
34  for (let key of Object.keys(book)) {
35      clonedBook[key] = book[key];
36  }
```

```
31  const book = {author: {firstname: "Th.",  surname: "Androutsos"}}
32  const clonedBook = {};
33
34  Object.assign(clonedBook, book)
35
36  console.log(clonedBook)
```

```
32  const book = {author: {firstname: "Th.",  surname: "Androutsos"}}
33  let clonedBook = {};
34
35  clonedBook = {...book}
36
37  console.log(clonedBook)
```

- 1. Με Object.keys
- 2. Με Object.assign
- 3. Πιο σύνηθες με spread operator

Object Serialization

- Το object **serialization** είναι η **μετατροπή σε JSON string** ενός object. Μετατρέπουμε το object state σε string από όπου μετά μπορούμε να ξανα-ανακτήσουμε το αρχικό object. Τα JSON strings έχουν ένα ειδικό, ωστόσο απλό, format)
- **Το JSON (JavaScript Object Notation) είναι ένα text-based format που χρησιμοποιείται κυρίως για τη μεταφορά δεδομένων σε εφαρμογές over web**. Έχει παρόμοια μορφή με τα JavaScript objects

JSON (1)

- Το JSON είναι η μετατροπή του object state σε string με ένα standard format όπου:
  - τα **keys είναι strings** μέσα σε double quotes
  - τα **values είναι JSON Types** (string, number, boolean, null, object, array). Όλα τα JSON string values πρέπει να είναι μέσα σε double quotes

```
let jsonStr = { "firstname": "Th.", "lastname": "Androutsos" }
```

JSON (2)

- Ο τύπος JSON number περιλαμβάνει όπως και ο JavaScript number δεκαδικούς αριθμούς (συμπεριλαμβανομένων ακεραίων) ενώ ο boolean είναι true/false

```
let jsonStr = { "firstname": "Th.", "lastname": "Androutsos" }
let jsonPoint = {"x": 1, "y": 1.5, isPoint2D: true}
```

- Κατ' αναλογία αναπαρίστανται και οι σύνθετοι τύποι, objects, arrays

JSON (3)

- Το **JSON object** είναι ένα non-instantiable utility class με static methods όπως **JSON.stringify()** που μετατρέπει ένα JS object σε JSON string και την **JSON.parse()** που μετατρέπει ένα JSON string πίσω σε JS object

```
1   const teacher = { id: 1, firstname: "Th.", lastname: "Androutsos"}
2
3   let jsonTeacher = JSON.stringify(teacher)
4   console.log(jsonTeacher)
5
6   let jsTeacher = JSON.parse(jsonTeacher)
7   console.log(jsTeacher)
```

Παράδειγμα Store (1)


```
const store = `[{ "id":"pro43vog",
                  "fields":{ "company":"ikea",
                             "colors":["#f15025","#222"],
                             "featured":true,
                             "price":999,
                              "genre":"white-back chair"
                           }},
    {"id":"pro23vpr",
    "fields":{ "company":"ikea",
                  "colors":["#f15025","#222"],
                  "featured":true,
                  "price":888,
                  "genre":"black-back chair"
    }}
]`
```

- Έστω το json που αναπαριστά ένα store object, δηλαδή ένα πίνακα από objects
- Κάθε object έχει δύο properties, όπου το 2ο property είναι object

Παράδειγμα Store (2)


```
18  function fetchProducts() {
19      const products = JSON.parse(store)
20      return products
21  }
22
23  function displayProducts(products) {
24      products.map(product => {
25          let { fields } = product
26          let { company, price, genre } = fields
27          console.log(company, price, genre)
28      })
29  }
30
31  const fetchedProducts = fetchProducts()
32  displayProducts(fetchedProducts)
```

- Η fetchProducts() μετατρέπει από JSON format σε JavaScript objects με **JSON.parse**
- Εμφανίζουμε αφού διασχίσουμε με map και κάνουμε destruct το κάθε product

Object Methods (1)

- toString() – Κληρονομείται από Object.prototype. By default δεν επιστρέφει κάτι παρά τον τύπο του object. Πρέπει να υπερκαλυφθεί ώστε να επιστρέφει ένα string με το state του object. Καλείται αυτόματα οπουδήποτε πρέπει να μετατραπεί ένα object σε string (π.χ. όταν έχουμε concat με string, όχι όμως στην console.log μιας και πρόκειται για browser-based συνάρτηση)

```
42  const point = {
43      x: 1,
44      y: 2,
45      toString: function() {return `(${this.x}, ${this.y})` }
46  }
47
48  console.log(point.toString())
49
```

Object Methods (2)

- toLocaleString() – Localized String representation

```
1  const point = {
2      x: 1000.54,
3      y: 2000.55,
4      toString: function() {return `(${this.x}, ${this.y})` },
5      toLocaleString: function() {return `(${this.x.toLocaleString()} -- ${this.y.toLocaleString('en-US')})`}
6  }
7
8  console.log(point.toString())
9  console.log(point.toLocaleString())
```

Object Methods (3)

- valueOf() – convert object σε primitive, τυπικά σε number

```
1  const point = {
2      x: 1,
3      y: 2,
4      toString: function() {return `(${this.x}, ${this.y})` },
5      valueOf: function() {return Math.hypot(this.x, this.y).toFixed(2)}
6  }
7
8  console.log(point.valueOf())
```

Arrays of Objects


```
1   const contacts = [
2       { firstname: "Athan",
3         lastname: "Andr.",
4         phoneNumber: "2108203900"
5       },
6       {
7         firstname: "Anna",
8         lastname: "W.",
9         phoneNumber: "6914455670"
10      }
11  ]
```

- Έχουμε ορίσει τον πίνακα mobileContacts που αναπαριστά τις επαφές ενός κινητού τηλεφώνου
- Περιέχει δύο objects
- Το κάθε object έχει τρία πεδία: τα **firstname, lastname, phonenumber,** με αντίστοιχες τιμές

Insert into contacts


```
1  const contacts = []
2
3  function insertContact(firstname, lastname, phone) {
4      if (!firstname || !lastname || !phone) return
5      contacts.push({firstname, lastname, phone})
6  }
7
8  insertContact("Athan", "Andr", "123456789")
9  console.log(contacts)
```

- Η **insertContact()** με βάση τα ορίσματα δημιουργεί το αντικείμενο *contact* και το εισάγει στο τέλος του πίνακα με την *.push()* . Χρησιμοποιεί το shorthand form στον object initializer
- Όταν έχουμε {firstname: firstname, lastname: lastname, phone: phone} μπορούμε να απλοποιήσουμε στο {firstname, lastname, phone}

Delete from contacts


```
38  function remove(phoneNumber) {
39      let numberFound = contacts.find(c => c.phoneNumber === phoneNumber)
40      if (!numberFound) return
41
42      let index = contacts.findIndex(c => c.phoneNumber === phoneNumber)
43      contacts.splice(index, 1)
44  }
```

- Με find επιστρέφουμε τον phone number που αναζητούμε
- Με findIndex() επιστρέφουμε τον index που περιέχει την επαφή με το συγκεκριμένο τηλέφωνο
- Διαγράφουμε με splice

Update contacts


```
46 function replace(oldNumber, newNumber) {
47     let oldNumberFound = contacts.find(c => c.phoneNumber === oldNumber)
48     let newNumberFound = contacts.find(c => c.phoneNumber === newNumber)
49
50     if (!oldNumberFound || newNumberFound) return
51
52     contacts.forEach(c => {
53         if (c.phoneNumber === oldNumber) c.phoneNumber = newNumber
54     })
55 }
```

- Αφού ελέγξουμε αν υπάρχει το oldNumber και αν δεν υπάρχει το newNumber προχωράμε στην αλλαγή μέσα σε μία forEach

Traverse contacts


```
31  function traverse(mobileContacts) {
32      mobileContacts.forEach(c => console.log(c))
33  }
34
35  traverse(mobileContacts)
```

Student object


```
let student = {
    id: 1,
    lastname: "Androutsos",
    firstname: "Athan.",
    address: {street: "Patission", number: "76"},
}
```

Object Destructuring


```
16  let { id, lastname: last, firstname: first }  = student
17  let { address } = student
18  let { street, number } = address
19
20  console.log(id, last, first)
21  console.log( "Address" + street, number)
```

- Στο αριστερό μέρος της παράστασης έχουμε το ίδιο όνομα με το object property key μέσα σε {}. Η βασική ιδέα είναι ότι εξάγουμε properties κάτι πολύ χρήσιμο όταν έχουμε objects και τις αντιστοιχούμε σε μεταβλητές με το ίδιο ή διαφορετικό identifier.
- Για διαφορετικό όνομα δίνουμε όπως στο firstname: first, πρώτα το property μετά : και μετά το όνομα της μεταβλητής

Cloning – Shallow copy


```
11  let clonedStu = {...student}
12  console.log(clonedStu)
```

- Πρόκειται για shallow copy

Συναρτήσεις

- Υπάρχουν δύο βασικοί τρόποι να ορίσουμε μία συνάρτηση
  - Με function add(a, b) // Declaration
  - Με const add = function(a, b) // Expression
- Στην ES6 και μετά μπορούμε να ορίσουμε και με ένα ακόμα τρόπο, με arrow functions

Μέθοδοι

- Επίσης, μπορούμε να ορίσουμε methods μέσα σε objects με δύο τρόπους
- Σαν property, π.χ. add: function() {return this.a + this.b}
- Σε shorthand μορφή, add() {return this.a + this.b}

Function Declaration (1)


```
1   const point = {x: 0, y: 1, z: 2}
2
3   function printPoint(point) {
4       for (let c in point) {
5           console.log(`${c}: ${point[c]}`);
6       }
7   }
8
9   printPoint(point)
```

- Οι συναρτήσεις στην JavaScript θεωρούνται objects και το όνομά τους είναι δείκτης στον κώδικα της συνάρτησης
- Είναι επίσης first-class objects. Μπορούν να υπάρξουν οπουδήποτε μπορεί να υπάρξει μία μεταβλητή

Function Declaration (2)


```
1   const point1 = {x: 0, y: 1, z: 2}
2   const point2 = {x: 10, y: 5, z: 6}
3
4   function printPoint(point) {
5       for (let c in point) {
6           console.log(`${c}: ${point[c]}`);
7       }
8   }
9
10  function getDistance(p1, p2) {
11      let dx = p1.x - p2.x;
12      let dy = p1.y - p2.y;
13      let dz = p1["z"] - p2["z"];
14      return Math.sqrt(dx*dx + dy*dy + dz*dz);
15  }
16
17  printPoint(point1)
18  console.log(getDistance(point1, point2).toFixed(2))
```

- Η δήλωση συναρτήσεων στην JS είναι απλή
- Δεν υπάρχουν επιστρεφόμενοι τύποι, ούτε τύποι παραμέτρων
- Ωστόσο επιστρέφουμε τιμές κανονικά με return
- Αν το return δεν ακολουθείται από expression επιστρέφει undefined

Αναδρομή


```
1   function factorial(n) {
2       if (n <= 1) return 1
3       return n * factorial(n-1)
4   }
5
6   console.log(factorial(100))
```

- Η αναδρομή δουλεύει όπως στην Java. Η συνάρτηση καλεί τον εαυτό της με μικρότερο μήκος δεδομένων
- Κίνδυνος για stack overflow αν το βάθος της αναδρομής είναι μεγάλο

Function Expression


```
1   const square = function(x) {
2       return x*x;
3   }
4
5   console.log(square(10))
6
```

```
const factorial = function facto(n) {
    //if (n <= 1) return 1; else return n * fact0(n-1)
    return (n <= 1) ? 1 : n * facto(n-1)
}

console.log(factorial(4))
```

- Δεδομένου ότι οι συναρτήσεις στην JavaScript είναι first-class objects, το όνομα της συνάρτησης είναι δείκτης και μπορεί μία ανώνυμη ή επώνυμη συνάρτηση να εκχωρηθεί σε ένα δείκτη
- Μπορούμε να δηλώνουμε ανώνυμες συναρτήσεις και να τις εκχωρούμε σε μεταβλητές που λειτουργούν ως δείκτες (Η μόνη διαφορά με τις κανονικές συναρτήσεις εδώ είναι ότι δεν δουλεύει το hoisting, γιατί τα expressions αν κληθούν πριν δηλωθούν δεν μπορεί να γίνει το reference στο όνομα, αφού το όνομα εκχωρείτε στη συνέχεια)
- Το όνομα της συνάρτησης είναι προαιρετικό. Χρειάζεται ωστόσο αν έχουμε αναδρομή, για να μπορούμε να αναφερθούμε

Arrow functions (1)

- Η γενική μορφή των arrow functions είναι μία comma-separated list από παραμέτρους σε παρενθέσεις, ακολουθούμενες από το => (arrow), ακολουθούμενο από το function body μέσα σε curly braces { }

```
const add = (x, y) => { return x + y; };
```

- Αλλά οι arrow functions υποστηρίζουν και ένα ακόμα πιο compact syntax form. Αν το σώμα της συνάρτησης είναι ένα μόνο return τότε μπορούμε να κάνουμε omit το return keyword καθώς και τα curly braces, και να γράψουμε μόνο το expression που πρόκειται να επιστραφεί

```
const sum = (x, y) => x + y;
const hash = (n) => n
```

Arrow functions (2)

- Επιπλέον αν η arrow function έχει μία μόνο παράμετρο μπορούμε να κάνουμε omit τις παρενθέσεις

```
const poly = x => x*x + 2*x + 3;
```

- Αν ωστόσο η arrow function δεν έχει καθόλου τυπικές παραμέτρους πρέπει να γράψουμε τις κενές παρενθέσεις

```
const constFun = () => 42;
```

Callback Functions

- Η compact μορφή των arrow functions τις κάνει ιδανικές για να περνούν ως παράμετροι συναρτήσεων, να λειτουργούν δηλαδή ως callback functions

```
let filtered = [1, null, 2, 3].filter(x => x !== null); // filtered == [1,2,3]

let squares = [1, 2, 3, 4].map(x => x*x);
```

Nested Function


```
1   function hypotenuse(a, b) {
2       function square(n) {
3           return n * n;
4       }
5
6       return Math.sqrt(square(a) + square(b));
7   }
```

- Το σημαντικό εδώ είναι ότι η nested function έχει πρόσβαση στο scope της outer function, όπως εδώ η square έχει πρόσβαση στα a, b

Object Methods


```
1   const obj = {
2       rate: 1.2,
3       getTotalAmount: function(amount) {
4           return amount + amount * this.rate;
5       },
6       getDoubleRate(amount) {
7           return amount + amount * (this.rate * 2)
8       }
9   }
10
11  console.log(obj.getTotalAmount(1000))
12  console.log(obj.getDoubleRate(1000))
```

- Τα methods εντός των objects έχουν πρόσβαση στα properties του object όπως το rate, **μέσω του this**

Function parameters (1)

- Η JS δεν ελέγχει τους τύπους των παραμέτρων συναρτήσεων και ούτε και το πλήθος τους
- Αν θέλουμε ωστόσο μπορούμε να ελέγχουμε αν έχουμε λιγότερες ή περισσότερες παραμέτρους από αυτές που δηλώνονται στις τυπικές παραμέτρους

Function parameters (2)


```
1  function add(a, b) {
2      return a + b
3  }
4
5  console.log(add(3))
6
```

```
ts\cf4\testbed> node .\testmethods.js
NaN
```

- Η JavaScript δεν κάνει έλεγχο παραμέτρων

Έλεγχος παραμέτρων


```
1   function add(a, b) {
2       if (a && b) {
3           return a + b
4       }
5
6       return 0
7   }
8
9   console.log(add(1, 5))
```

```
9   function mul(a, b) {
10      a = a || 0
11      b = b || 0
12
13      return a * b
14  }
15
16  console.log(mul(1))
```

- Αντί για if μπορούμε να χρησιμοποιήσουμε το || με αυτόν τον ιδιωματικό τρόπο

Idiomatic || και &&


```
function mul(a, b) {
    a = a || 0
    b = b || 0

    return a * b
}
```

- Τα && και || είναι short circuit και επιστρέφουν όχι τιμή true ή false αλλά **την τιμή της μεταβλητής που σταματάει το short circuit ή διαφορετικά την τελευταία μεταβλητή**
- Στο || αν το a είναι truthy επιστρέφεται το a (η τιμή του a), αλλιώς το 0.

Optional params


```
1  function add(a=0, b=0)  {
2      return a + b
3  }
4
5  console.log(add())
6  console.log(add(1))
7  console.log(add(3, 6))
```

- Προαιρετικές παραμέτρους δίνουμε με εκχώρηση default τιμών στην επικεφαλίδα των παραμέτρων της συνάρτησης
- Είναι μία μορφή overloading

Rest and max


```
1   function max(...rest) {
2       let maxValue = -Infinity;
3
4       for(let n of rest) {
5           if (n > maxValue) {
6               maxValue = n;
7           }
8       }
9
10      return maxValue;
11  }
12
13  console.log(max(1, 10, 100, 2, 3, 1002, 4, 5, 6))  // => 1002
14
```

- Τρεις τελείες πριν τη μεταβλητή, πρόκειται για rest (varArgs στην Java). Μπορούμε να περάσουμε μηδέν, μία ή περισσότερες παραμέτρους. Υλοποιείται με πίνακα. Πρέπει να είναι η τελευταία τυπική παράμετρος

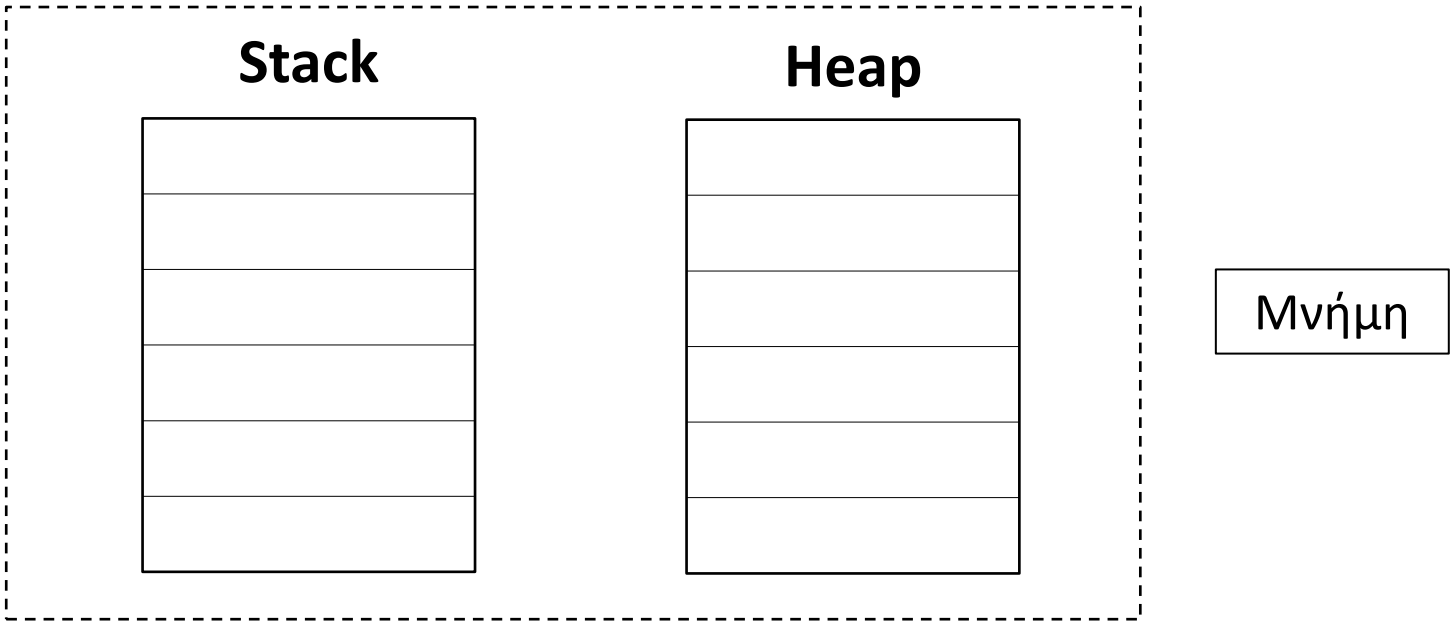
Spread operator and Math


```
1   console.log(max(1, 2, 3, 4))
2
3   function max(...nums) {
4       return Math.max(...nums)
5   }
```

- Όπως έχουμε ξαναδεί με Math.max μπορούμε να πάρουμε τον μέγιστο μίας λίστας στοιχείων ή ενός πίνακα (με spread operator)

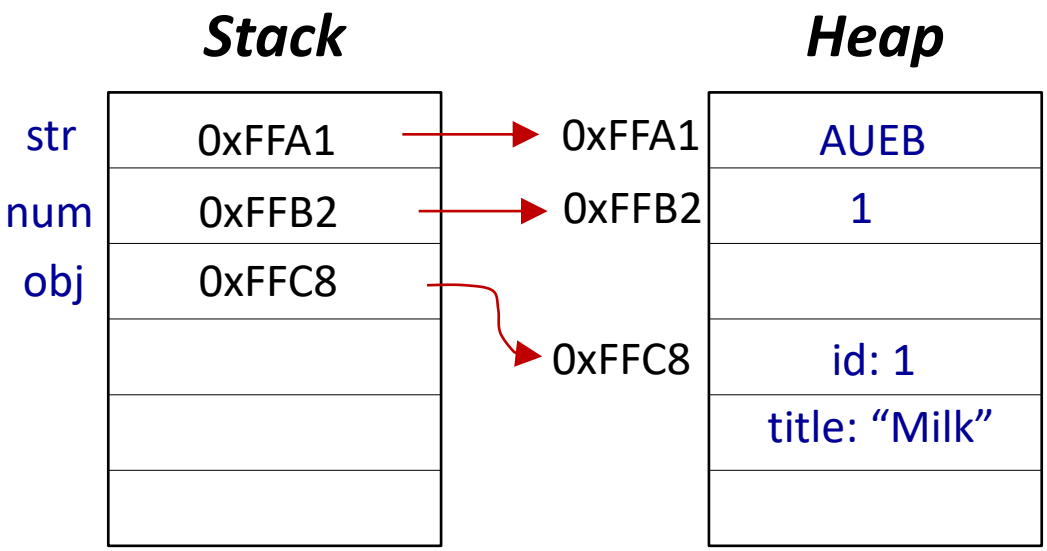
Διαχείριση Μνήμης

- Η JavaScript (όπως και η Java και C# και άλλες γλώσσες) χωρίζουν το χώρο μνήμης που διαχειρίζονται σε δύο μέρη: στη στοίβα (**stack**) και στο σωρό (**heap**)



Δυναμική Δέσμευση

- Η **στοίβα (stack)** χρησιμοποιείται για δέσμευση χώρου για references. Οι παράμετροι και τα local variables των συναρτήσεων αποθηκεύονται έξω από το stack
- Ο **Σωρός (heap)** χρησιμοποιείται για δέσμευση χώρου στη μνήμη για όλες τις μεταβλητές:




```
let str = “AUEB”;
let num = 1;
let obj = {id: 1, title: “Milk”};
```

- Οι μεταβλητές str, num, obj ορίζονται στο stack και περιέχουν τις διευθύνσεις μνήμης των τιμών, είναι δηλαδή στην πραγματικότητα references (αναφορές) ή αλλιώς δείκτες (pointers) προς τις πραγματικές τιμές, που είναι στο heap
- Είναι επομένως άλλο πράγμα οι δείκτες και άλλο οι τιμές στις οποίες δείχνουν

Πέρασμα παραμέτρων σε συναρτήσεις

- Υπάρχουν τρεις βασικοί τρόποι περάσματος παραμέτρων σε συναρτήσεις:
  - **By value**, όπου οι τιμές περνάνε ως είσοδος μόνο και οι όποιες αλλαγές δεν ισχύουν έξω από τη συνάρτηση
  - **By reference**, όπου ως είσοδο περνάμε δείκτες και οι όποιες αλλαγές στις τιμές που δείχνουν οι δείκτες ισχύουν κανονικά έξω από τη συνάρτηση. Δείκτες έχει βασικά η γλώσσα C
  - **By value and reference**, όπου ως είσοδο περνάμε αναφορικές μεταβλητές που λειτουργούν ως δείκτες και όποιες αλλαγές γίνουν στις τιμές που δείχνουν οι αναφορικές μεταβλητές, ισχύουν έξω από τη συνάρτηση. Αν αλλάξουμε την τιμή της αναφορικής μεταβλητής αυτής καθαυτής δεν ισχύει η αλλαγή έξω από τη συνάρτηση

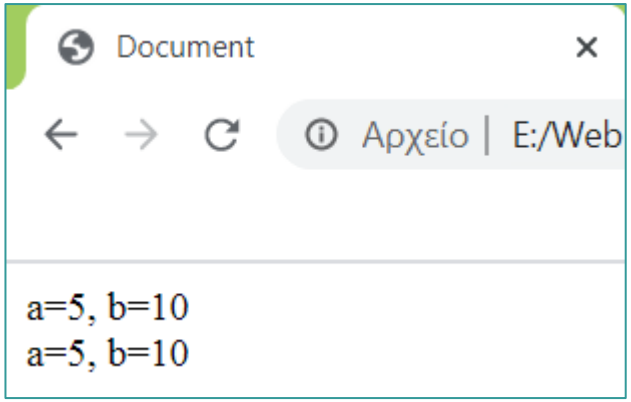
JavaScript

- Στην JavaScript όλα περνάνε κατά τιμή. Αυτό που περνάει δηλαδή κατά τιμή είναι η αναφορά (reference)
- Επομένως, τα ίδια τα references δεν αλλάζουν, με την έννοια ότι παραμένουν τα ίδια μετά την έξοδο από τη συνάρτηση

Call by value (1)

testbed > swap.html > ...

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device
6       <title>Call by value</title>
7   </head>
8   <body>
9       <script>
10          let a = 5;
11          let b = 10;
12
13          function swap (a, b) {
14              tmp = a;
15              a = b;
16              b = tmp;
17          }
18
19          document.write(`a=${a}, b=${b} <br>`);
20          swap(a, b);
21          document.write(`a=${a}, b=${b}`);
22
23      </script>
24  </body>
25  </html>
```



- Η **swap(a, b)** αντιστρέφει αμοιβαία τις τιμές των τυπικών παραμέτρων a, b
- Παρατηρήστε ωστόσο ότι όταν περνάμε ως παραμέτρους τις πραγματικές παραμέτρους a = 5, και b = 10 δεν γίνεται η αμοιβαία ανταλλαγή
- Γιατί συμβαίνει αυτό; Δείτε στην επόμενη διαφάνεια

Call by value (2)


```
function swap (a, b) {
    tmp = a;
    a = b;
    b = tmp;
}
```

- Τα a, b είναι τυπικές παράμετροι της swap
- Επειδή οι τυπικές παράμετροι είναι *τοπικές μεταβλητές* των συναρτήσεων **όποιες αλλαγές γίνουν σε αυτές δεν ισχύουν έξω από τη συνάρτηση**
- Λέμε ότι οι παράμετροι συναρτήσεων στην JavaScript περνάνε πάντα **by value**, δηλαδή **μόνο ως είσοδος** τιμών και όχι ως έξοδος

Call by value & reference (1)

- Όπως αναφέραμε οι μεταβλητές είναι στην πραγματικότητα αναφορές (references) προς τις πραγματικές τιμές
- Επομένως όταν περνάμε ως παράμετρο σε μια συνάρτηση μια αναφορική μεταβλητή, αυτό που περνάει ως είσοδος είναι η διεύθυνση μνήμης στο heap των πραγματικών τιμών

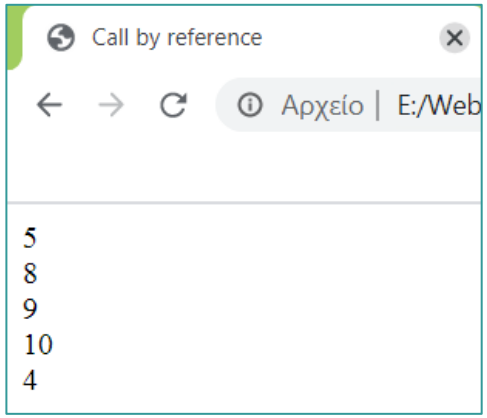
Call by value and reference (2)


```
<script>
    function roundGrades(grades) {
        for (let i = 0; i < grades.length; i++) {
            grades[i] = Math.round(grades[i]);
        }
    }

    let studentGrades = [4.7, 8.3, 9.1, 9.7, 3.5];

    roundGrades(studentGrades);

    for (let grade of studentGrades) {
            document.write(`${grade} <br>`);
    }
</script>
```



- Η συνάρτηση *roundGrades()* παίρνει ως είσοδο ένα πίνακα
- Αλλάζει τις τιμές του πίνακα, στρογγυλοποιώντας με την *Math.round()*
- Οι αλλαγές ισχύουν και έξω από τη συνάρτηση αφού η grades δείχνει σε object όπως είναι ο πίνακας και επομένως αυτό που αλλάζει είναι οι τιμές του πίνακα όχι η αναφορά στον πίνακα η οποία περνάει κατά τιμή

First Class Functions

- Στην JavaScript οι συναρτήσεις είναι *First Class Functions*, δηλαδή μπορούν να χρησιμοποιηθούν όπως οι μεταβλητές της γλώσσας
- Δηλαδή μπορούν να *εκχωρηθούν σε μεταβλητές ή* να περάσουν ως τυπικές παράμετροι (*callback functions*) σε άλλες συναρτήσεις και γενικότερα να χρησιμοποιηθούν όπως μία μεταβλητή, όπως θα δούμε στα επόμενα

Ανώνυμες Συναρτήσεις (1)

- Ανώνυμες (anonymous functions) ονομάζονται οι συναρτήσεις που ορίζονται χωρίς όνομα, ως εξής: **function() { ..}**
- Η χρησιμότητά τους έγκειται στο ότι μπορούν να χρησιμεύσουν απλά ως μπλοκ κώδικα ιδιαίτερα όταν είναι να χρησιμοποιηθούν μία μόνο φορά ή μπορούν να εκχωρούνται σε μεταβλητές οπότε μπορούν να λαμβάνουν έμμεσα όνομα με ευέλικτο τρόπο

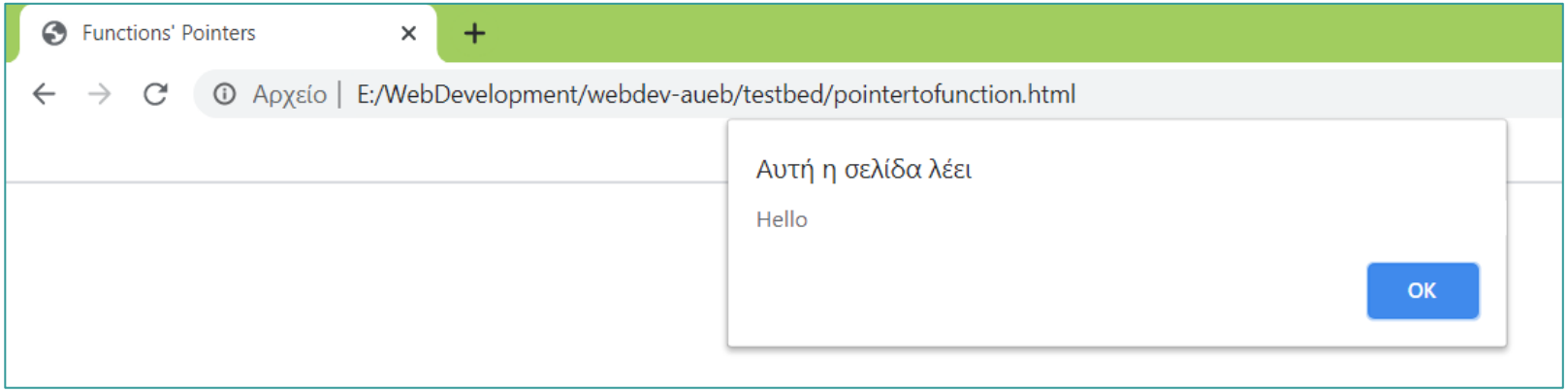
Ανώνυμες Συναρτήσεις (2)

- Μία ανώνυμη συνάρτηση είναι ένα μπλοκ κώδικα στη μνήμη προς το οποίο όμως μπορεί να ‘δείχνει’ μία αναφορική μεταβλητή
- Μπορούμε δηλαδή να εκχωρούμε σε μία μεταβλητή την ανώνυμη συνάρτηση και επομένως να ορίσουμε ένα δείκτη προς τη συνάρτηση που θα είναι και το όνομα της συνάρτησης, π.χ.
- *let helloAlert = function() {alert(“hello”); }*
- Μετά μπορούμε να καλέσουμε:
- *helloAlert();*

Ανώνυμες Συναρτήσεις (3)


```
 8   <body>
 9       <script>
10           let helloAlert = function() {
11               alert('Hello');
12           }
13
14           helloAlert();
15       </script>
16   </body>
```

- Το **helloAlert** (γραμμή 10) είναι μεταβλητή στην οποία εκχωρούμε τη συνάρτηση
- Λειτουργεί ως *δείκτης* αλλά και ως *όνομα* της συνάρτησης με το οποίο μπορούμε να την καλούμε (βλ. γραμμή 14)



Callback functions

- Υπάρχουν περιπτώσεις που μία συνάρτηση για να επιτελέσει το έργο της εξαρτάται από μία άλλη συνάρτηση που περνάει ως παράμετρος (callback function)
- Callback functions ονομάζονται οι συναρτήσεις που περνάνε ως παράμετροι / ορίσματα σε άλλες συναρτήσεις
- Είναι σύνηθες να περνάμε ως callback function μία anonymous function, ιδιαίτερα όταν θέλουμε να την χρησιμοποιήσουμε μόνο μια φορά

Arrow functions

- Πρόκειται για μία πιο *σύντομη μορφή* ορισμού και κλήσης ανώνυμων συναρτήσεων
- Μορφή: ***(λίστα παραμέτρων) => {κώδικας}***
  Αν ο κώδικας έχει μόνο μία εντολή δεν είναι απαραίτητα τα {}. Σε αυτή την περίπτωση, το return υπονοείται
- Π.χ.
```
let add = (a, b) => a + b;
add(3, 5);     // 8
```
- *Π.χ. function sayHello(() => console.log('Hello'));*
- Π.χ.
```
function controlService((amount) => {
    paymentService(amount); console.log("Payment OK")})
```

Closures (1)

- Υπάρχουν περιπτώσεις που μία συνάρτηση περιέχει στο σώμα της τον ορισμό μιας ή περισσοτέρων άλλων συναρτήσεων, δηλαδή σαν να υπάρχει μία σχέση ιδιοκτησίας
- Οι εσωτερικές συναρτήσεις (inner functions) έχουν πρόσβαση στις μεταβλητές της outer function ακόμα και όταν η outer function έχει τελειώσει την εκτέλεσή της
- Είναι μία προσομοίωση των κλάσεων αλλά με συναρτησιακό τρόπο

Closures (2)

- Επομένως ο χώρος μνήμης της outer function διατηρείται εφόσον υπάρχει μία inner function, δηλαδή οι μεταβλητές της εξωτερικής συνάρτησης παραμένουν στη θέση τους ή με άλλα λόγια είναι ‘κλειστές’
- Ο χώρος μνήμης της εξωτερικής συνάρτησης μαζί με τις εσωτερικές συναρτήσεις συνιστά ένα **closure**

Closure


```
8   <body>
9       <script>
10          function bookTheaterSeat(seatNumber) {
11              let seats = [{seat: 'A1', price: 10, booked: true}, {seat: 'A2', price: 20, booked: true}];
12              let found = false;
13
14              return function book() {
15                  for (let seat of seats) {
16                      if (seat['seat'] == seatNumber) {
17                          found = true;
18                          if (seat['booked'] == false) {
19                              seat['booked'] = true;
20                              alert('Seat ' + seatNumber + ' booked');
21                          } else {
22                              alert('Seat ' + seatNumber + ' is already booked');
23                          }
24                      }
25                  }
26
27                  if (!found) {
28                      alert('Seat ' + seatNumber + ' not found');
29                  }
30              }
31          }
32
33          let bookA2Seat = bookTheaterSeat('A3');
34          bookA2Seat();
35      </script>
36  </body>
```

- Εδώ η bookTheaterSeat() περιέχει την book() και κάνει return την book() – προσοχή κάνει return όλη τη συνάρτηση, όχι μια τιμή. Η book() στη συνέχεια εκχωρείται στην bookA2Seat καλώντας την bookTheaterSeat('A2') και έχει πρόσβαση στις μεταβλητές της bookTheaterSeat() δηλαδή στις seats, found και seatNumber ακόμα και όταν η bookTheaterSeat('A2') έχει τελειώσει την εκτέλεσή της (γραμμή 29)

Set (1)

- Το Set είναι μία δομή δεδομένων με κύριο χαρακτηριστικό ότι δεν επιτρέπονται διπλότυπα
- **Μπορούμε να μετατρέψουμε σε πίνακα με spread**
- Μπορούμε κατά συνέπεια να εφαρμόσουμε και Array methods όπως forEach, filter, map, κλπ.

Set (2)

- Αρχικοποιείται με new Set() ή new Set(iterable)
- Παρέχονται οι μέθοδοι
  - add()
  - delete()
  - has(element)
  - Καθώς και το property **.size** που δίνει το μέγεθος του Set

Set (3)

testbed > JS set.js > ...

```
1  let bag = new Set()
2  bag.add('Oranges')
3  bag.add('Apples')
4  bag.add('Oranges') // Set does not permit duplicates
5  bag.delete('Oranges')
6  bag.add('Honey')
7
8  if (bag.has('Apples')) {
9      console.log('Has Apples')
10 }
11
12 console.log(`Bag has size: ${bag.size}`)
13 console.log('Items:', bag)
```

Unique cities

testbed > JS mapSet.js > ...

```
let store = [{product: 'Apples', city: "Lamia"},
             {product: 'Oranges', city: "Athens"},
             {product: 'Milk', city: "Volos"},
             {product: "Honey", city: "Athens"}]


let cities = ['all', ...new Set(store.map((product) => product.city))]

console.log(cities)
```

- Έστω ένα πίνακας από objects που έχουν και ένα property city
- Μπορούμε να εξάγουμε με map, να πάρουμε τα unique με Set, και να μετατρέψουμε σε πίνακα

Filtered Cities

testbed > JS mapSet.js > ...

```
1  let store = [{product: 'Apples', city: "Lamia"},
2               {product: 'Oranges', city: "Athens"},
3               {product: 'Milk', city: "Volos"},
4               {product: "Honey", city: "Athens"}]
5
6
7  let cities = ['all', ...new Set(store.map((product) => product.city))]
8
9  let filtered = cities.filter(city => city.startsWith('V'))
10 console.log(filtered)
```

- Το ίδιο όπως πριν, μόνο που έχει προστεθεί και ένα filter

JS Modules

- Τα modules είναι ανεξάρτητες μονάδες κώδικα όπου όλες οι **δηλώσεις μεταβλητών και συναρτήσεων είναι private μέσα στο Module** και μπορούν να γίνουν ρητά export ώστε κάποιο πρόγραμμα JS ή άλλο Module να τις κάνει import
- Αν δεν έχουμε modules όλες οι δηλώσεις μεταβλητών έξω από τις συναρτήσεις θεωρούνται global και όλες οι συναρτήσεις έχουν πρόσβαση σε αυτές
- Επίσης, οι δηλώσεις var και συναρτήσεων θεωρούνται μέρος του global object και μπορούν να κληθούν ως window.<variable-name ή function-name>

Modules (1)

testbed > modules > JS main.js > sub

```
1  export default function add(a, b) {
2      return a + b
3  }
4
5  export function sub(a, b) {
6      return a - b
7  }
8
9  export function mul(a, b) {
10     return a * b
11 }
12
13 export function div(a, b) {
14     return a / b
15 }
```

- Τα modules είναι βιβλιοθήκες συναρτήσεων
- Ο λόγος ύπαρξής τους είναι ο διαμοιρασμός (export), ενώ αρχιτεκτονικά επιτυγχάνεται modularization του κώδικα
- Άλλα .js αρχεία μπορούν να κάνουν import όποιες συναρτήσεις χρειάζονται και να τις χρησιμοποιούν
- Μπορεί να υπάρχει **μόνο ένα export default** και πολλά named exports ή μόνο named exports

Modules (2)

testbed > modules > JS main.js > ...

```
1   export default function add(a, b) {
2       return a + b
3   }
4
5   function sub(a, b) {
6       return a - b
7   }
8
9   function mul(a, b) {
10      return a * b
11  }
12
13  function div(a, b) {
14      return a / b
15  }
16
17  export { sub, mul, div }
```

- Μπορεί να υπάρχει και αυτή η μορφή με ένα **export στο τέλος**
- Στα named exports κάθε function διατηρεί το όνομά της στο importing module
- Στο default export, το importing module μπορεί να δώσει άλλο όνομα στο default function

Modules (3)

testbed > modules > JS calc.js > ...

```
1   import add, { sub, mul, div} from './main.js'
2
3   let mySum = add(3, 5)
4   let mySub = sub(7, 1)
5   let myMul = mul(3, 7)
6   let myDiv = div(4, 3)
7
8   console.log("sum", mySum)
9   console.log("sub", mySub)
10  console.log("mul", myMul)
11  console.log("div", myDiv.toFixed(2))
```

- Με *import* κάνουμε import το default module και τα named modules με το syntax όπως φαίνεται παραπάνω

Browser

testbed > modules > calc.html > html

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-widt
6       <title>Document</title>
7   </head>
8   <body>
9
10      <script type="module" src="./calc.js"></script>
11  </body>
12  </html>
```

- Τα modules φορτώνονται με type="module" στον browser, ώστε να γνωρίζει ο browser ότι το script χρησιμοποιεί ECMAScript module syntax, με import και export

Ασκήσεις (1)

1. Δεδομένου ενός πίνακα αριθμών, χρησιμοποιήστε τη συνάρτηση map για να δημιουργήσετε έναν νέο πίνακα όπου κάθε αριθμός διπλασιάζεται
2. Δεδομένου ενός πίνακα αριθμών, χρησιμοποιήστε τη συνάρτηση filter για να δημιουργήσετε έναν νέο πίνακα που περιλαμβάνει μόνο τους ζυγούς αριθμούς
3. Δεδομένου ενός πίνακα αριθμών, χρησιμοποιήστε τη συνάρτηση some για να ελέγξετε εάν τουλάχιστον ένας αριθμός είναι θετικός
4. Δεδομένου ενός πίνακα αριθμών, χρησιμοποιήστε τη συνάρτηση every για να ελέγξετε εάν όλοι οι αριθμοί είναι θετικοί

Ασκήσεις (2)

1. Δεδομένου ενός πίνακα αντικειμένων, χρησιμοποιήστε τη συνάρτηση filter για να δημιουργήσετε έναν νέο πίνακα που περιλαμβάνει μόνο χρήστες κάτω των 30 ετών
2. Στη συνέχεια χρησιμοποιήστε τη συνάρτηση map για να δημιουργήσετε έναν νέο πίνακα που περιέχει μόνο τα ονόματα.

```
const people = [
  { name: 'Alice', age: 30 },
  { name: 'Bob', age: 25 },
  { name: 'Charlie', age: 35 },
];
```

Ασκήσεις (3)

- Δημιουργήστε ένα module με το όνομα **greeting.js**. Μέσα στο module, εξάγετε ένα default function που κάνει log ένα μήνυμα χαιρετισμού
- Σε ένα άλλο αρχείο **main.js** εισάγετε και χρησιμοποιήστε την default συνάρτηση για να εμφανίσετε ένα "Hello World".
- Σε ένα αρχείο HTML, χρησιμοποιήστε το main.js