

**ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ**



**ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS**

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Typescript και Angular

Βασικά χαρακτηριστικά του πρότυπου ECMAScript 6

Χριστόδουλος Φραγκουδάκης



ECMAScript → Typescript → Angular

Το Angular Framework είναι βασισμένο στη γλώσσα προγραμματισμού Typescript που είναι ένα υπερσύνολο της προδιαγραφής ECMAScript 6 (ES6) με επιπλέον δυνατότητες.

Η Typescript εμπλουτίζει τη Javascript:

- με στατικό έλεγχο τύπων
- και με κλάσεις που επιτρέπουν OOP προγραμματισμό

Το Angular Framework αξιοποιεί πολλές δυνατότητες της ES6:

- Arrow functions, κλάσεις, template literals,
- Interfaces, Decorators, κτλ

Ξεκινάμε με τις βασικές έννοιες των αντικειμένων στη Javascript από το πρότυπο ECMAScript 1



Το μοντέλο των αντικειμένων έως το πρότυπο ECMAScript 5

- Παρέχεται από τη Javascript σαν ένας τρόπος αναπαράστασης και χειρισμού δεδομένων
- Τα πάντα είναι αντικείμενα ή αντιμετωπίζονται ως αντικείμενα (συμβολοσειρές, αριθμοί, ...)
- Βασίζεται στην αναφορά σε **πρωτότυπα αντικείμενα** που επιτρέπουν την κληρονομικότητα ιδιοτήτων και μεθόδων από άλλα αντικείμενα: **ένα νέο αντικείμενο κληρονομεί τις ιδιότητες και τις μεθόδους του πρωτότυπου αντικειμένου**
- Είναι ιεραρχικό με δομή δέντρου που στη ρίζα του έχει το αντικείμενο `Object` που είναι το βασικό πρωτότυπο αντικείμενο που κληρονομούν όλα τα άλλα αντικείμενα
- Το `Object` έχει ενσωματωμένες ιδιότητες και μεθόδους (`toString`, `valueOf()`, ...) που μπορούν να χρησιμοποιηθούν από όλα τα αντικείμενα που κληρονομούν από αυτό
- Τα αντικείμενα είναι **μεταβλητά (mutable)**, δηλαδή μπορούμε να αλλάξουμε τα χαρακτηριστικά και τις μεθόδους **δυναμικά** κατά τη διάρκεια της εκτέλεσης



Το καθολικό αντικείμενο `Object`

Υπάρχει σε όλες τις υλοποιήσεις της Javascript και παρέχει μεθόδους για τη δημιουργία (**κατασκευάστρια συνάρτηση, *constructor function***) και το χειρισμό αντικειμένων

Οι κατασκευάστριες συναρτήσεις καλούνται με τη χρήση του `new` και στο παράδειγμα δημιουργείται ένα κενό αντικείμενο.

```
const myObj = new Object();  
myObj.foo = "bar";  
  
console.log(myObj.foo); // Τυπώνει "bar"
```

Στη συνέχεια προσθέτουμε την ιδιότητα `foo` με τη σύνταξη `myObj.foo` (dot notation)

Η `Object` παρέχει διάφορες **στατικές μεθόδους** που επιτρέπουν το χειρισμό των αντικειμένων

```
myObj.baz = "qux";  
const keys = Object.keys(myObj);  
console.log(keys); // Τυπώνει ["foo", "baz"]
```

Στο παράδειγμα η `Object.keys()` επιστρέφει πίνακα με τα ονόματα των χαρακτηριστικών του αντικειμένου



Κυριολεκτική αναπαράσταση αντικειμένου

- Η κυριολεκτική αναπαράσταση δημιουργεί κατευθείαν ένα αντικείμενο με συγκεκριμένες ιδιότητες και τιμές
- Είναι συντόμευση αντί της χρήσης κατασκευάστριας συνάρτησης
- Στο παράδειγμα δημιουργούμε κενά αντικείμενα και με τους δύο τρόπους
- Πρόκειται για ισοδύναμους τρόπους δημιουργίας κενού αντικειμένου

```
const a = {}; // Κυριολεκτική αναπαράσταση
const b = new Object(); // Κατασκευάστρια συνάρτηση

console.log(typeof a); // Τυπώνει "object"
console.log(typeof b); // Τυπώνει "object"
console.log(a instanceof Object); // Τυπώνει true
console.log(b instanceof Object); // Τυπώνει true
console.log(Object.getPrototypeOf(a) === Object.prototype);
// Τυπώνει true
console.log(Object.getPrototypeOf(b) === Object.prototype);
// Τυπώνει true
```

- Τα αντικείμενα `a` και `b` είναι και τα δύο στιγμιότυπα του `Object`
- Το πρωτότυπο και των δύο αντικειμένων είναι το πρωτότυπο του `Object`



Βασικές στατικές μέθοδοι του Object

- Η `Object.assign()` αντιγράφει ζευγάρια χαρακτηριστικών-τιμής από ένα αντικείμενο σε άλλο αντικείμενο

```
let copiedObj = Object.assign(myObj, { id: 2 });  
console.log(copiedObj.valueOf());  
// Τυπώνει { foo: 'bar', baz: 'qux', id: 2 }
```

- Η `Object.entries()` μετατρέπει τα αντικείμενα σε πίνακες

```
const entries = Object.entries(copiedObj);  
console.log(entries);  
// Τυπώνει [ [ 'foo', 'bar' ], [ 'baz', 'qux' ], [ 'id', 2 ] ]
```

- Η `Object.fromEntries()` λειτουργεί αντίστροφα της `Object.entries()`

```
const revonvertedObj = Object.fromEntries(entries);  
console.log(revonvertedObj);  
// Τυπώνει { foo: 'bar', baz: 'qux', id: 2 }
```

- Η `Object.keys()` επιστρέφει ένα πίνακα με τα ονόματα των χαρακτηριστικών του αντικειμένου

```
const keys = Object.keys(copiedObj);  
console.log(keys);  
// Τυπώνει [ 'foo', 'baz', 'id' ]
```



Συναρτήσεις και αντικείμενα

- Στη Javascript οι συναρτήσεις και τα αντικείμενα είναι στενά συνδεδεμένες έννοιες. Οι συναρτήσεις μπορεί να θεωρηθούν σαν ένας **ειδικός τύπος αντικειμένου**
- Οι συναρτήσεις μπορεί να έχουν χαρακτηριστικά και μεθόδους όπως κάθε άλλο αντικείμενο της γλώσσας, π.χ. το χαρακτηριστικό `length` που υποδηλώνει τον αριθμό των ορισμάτων ή η μέθοδος `call()` που επιτρέπει την κλήση της συνάρτησης με συγκεκριμένο `this`, κτλ
- Οι συναρτήσεις και τα αντικείμενα έχουν την ιδιότητα `prototype` που υποδηλώνει το αντικείμενο από το οποίο κληρονομούνται ιδιότητες και μέθοδοι
- Η ιδιότητα `prototype` μας επιτρέπει να ορίσουμε ένα σύνολο κοινών ιδιοτήτων και μεθόδων που μπορούν να χρησιμοποιηθούν σε όλα τα στιγμιότυπα των αντικειμένων έτσι ώστε να μειώνεται η επανάληψη και ο κώδικας να είναι αποδοτικός



Η έννοια του `this`

Η λέξη κλειδί `this` αναφέρεται στα "συμφραζόμενα" της εκτέλεσης μιας συνάρτησης. Είναι αναφορά στο αντικείμενο της κλήσης της συνάρτησης. *Η τιμή του `this` καθορίζεται από τον τρόπο κλήσης της συνάρτησης και όχι από το σημείο που ορίζεται η συνάρτηση.*

Καθολικό `this`

Έξω από οποιοδήποτε αντικείμενο αναφέρεται στο *καθολικό αντικείμενο*

```
function globalFunction() {  
  console.log(this.firstName);  
}  
  
globalFunction(); // Τυπώνει undefined
```

Μέθοδος αντικειμένου

Αναφέρεται στο αντικείμενο που μέσω του οποίου καλείται η μέθοδος

```
const obj = {  
  firstname: "John",  
  what: globalFunction,  
};  
  
obj.what(); // Τυπώνει John
```




Η έννοια του `this`

Μέθοδοι συναρτήσεων `call()`, `apply()`, ...

Σε αυτές τις κλήσεις η τιμή του `this` περνά σαν παράμετρος

```
const obj1 = { x: 10 };
const obj2 = { x: 20 };

function printX() {
  console.log(this.x);
}

printX.call(obj1); // Το 'this' είναι το 'obj1', άρα τυπώνει 10
printX.call(obj2); // Το 'this' είναι το 'obj2', άρα τυπώνει 20
```

Κατασκευάστρια συνάρτηση

Αναφέρεται στο νέο αντικείμενο που δημιουργεί η κλήση με το `new`

```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

const person1 = new Person("John", "Doe"); // Αναφέρεται στο person1
```

Χειριστές γεγονότων του DOM

Στους χειριστές γεγονότων αναφέρεται στο στοιχείο του DOM που συνέβη το γεγονός

```
button.addEventListener("click", function () {
  console.log(this); // Αναφέρεται στο συγκεκριμένο κουμπί
});
```



To `this` μέσα και έξω από το REPL (Read-Eval-Print Loop) prompt του Node.js

```
whatisthis.js
File Edit View
console.log(this)
Ln 1, Col 18 | 100% | Windows (CRLF) | UTF-8
```

```
~/Workspace/Coding Factory/2023/examples/ecmascript
christodoulos@laptop ~/Workspace/Coding Factory/2023/examples/ecmascript
$ node whatisthis.js
{}
christodoulos@laptop ~/Workspace/Coding Factory/2023/examples/ecmascript
$
```

```
~/Workspace/Coding Factory/2023/examples/ecmascript
christodoulos@laptop ~/Workspace/Coding Factory/2023/examples/ecmascript
$ node
Welcome to Node.js v18.15.0.
Type ".help" for more information.
> console.log(this)
<ref *1> Object [global] {
  global: [Circular *1],
  queueMicrotask: [Function: queueMicrotask],
  clearImmediate: [Function: clearImmediate],
  setImmediate: [Function: setImmediate] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  structuredClone: [Function: structuredClone],
  clearInterval: [Function: clearInterval],
  clearTimeout: [Function: clearTimeout],
  setInterval: [Function: setInterval],
  setTimeout: [Function: setTimeout] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  atob: [Function: atob],
  btoa: [Function: btoa],
  performance: Performance {
    nodeTiming: PerformanceNodeTiming {
      name: 'node',
      entryType: 'node',
      startTime: 0,
      duration: 11831.194599986076,
      nodeStart: 3.3396999835968018,
      v8Start: 7.098600000143051,
      bootstrapComplete: 32.642100006341934,
      environment: 17.226700007915497,
      loopStart: 55.38299998641014,
      loopExit: -1,
      idleTime: 11702.515
    },
    timeOrigin: 1679130541629.75
  },
  fetch: [AsyncFunction: fetch]
}
undefined
>
```

Όταν το πρόγραμμα εκτελείται από τη γραμμή εντολών τότε "περιτυλίγεται" (wrapped) σε μια συνάρτηση που το αντικείμενο που επιδρά είναι κενό:

```
(function (exports, require, module, __filename, __dirname) {
  // Ο κώδικας βρίσκεται στην πραγματικότητα εδώ
});
```

Εντός του REPL prompt δεν υπάρχει αυτή η συνάρτηση και το αντικείμενο που αναφέρεται το `this` είναι το καθολικό αντικείμενο `global`.



Τα καθολικά αντικείμενα `global` και `window`

Μέσα στο web browser το καθολικό αντικείμενο είναι το `window`, ενώ στο Node.js REPL prompt είναι το `global`. *Όταν βρισκόμαστε έξω από οποιαδήποτε συνάρτηση ή αντικείμενο, το `this` αναφέρεται στο καθολικό αντικείμενο.*

```
Command Prompt - nc x + v - □ ×
C:\Users\christodoulos>node
Welcome to Node.js v18.15.0.
Type ".help" for more information.
> console.log(this === global)
true
undefined
> |
```

```
about:blank x + v - □ ×
about:... |
Elements Console Sources Network >>
top | Filter Default levels No Issues
> console.log(this === window)
true VM135:1
< undefined
> |
```



Κυριολεκτικές αναπαραστάσεις (Object Literals) αντικειμένων στο πρότυπο ES5

- Μπορούμε να δημιουργήσουμε αντικείμενα αν εσωκλείσουμε ζευγάρια key-value ανάμεσα στις αγκύλες `{}`
- Στο παράδειγμα χρησιμοποιούμε ένα object literal για να δημιουργήσουμε ένα αντικείμενο με συγκεκριμένα χαρακτηριστικά και μεθόδους
- Όταν καλείται η συνάρτηση `greet`, **η αναφορά του `this` εξαρτάται από τον τρόπο της κλήσης της συνάρτησης**

```
const obj = {  
  name: "John",  
  age: 30,  
  greet: function () {  
    console.log(`Hello, my name is ${this.name}.`);  
  },  
};  
obj.greet(); // Τυπώνει "Hello, my name is John."
```

- Αν καλείται σαν μέθοδος αντικειμένου τότε αναφέρεται στο ίδιο το αντικείμενο
- Αν κληθεί μετά από ανάθεση σε καθολική μεταβλητή τότε αναφέρεται στο καθολικό αντικείμενο

```
const pgreet = person.greet;  
pgreet(); // Τυπώνει "Hello, my name is undefined."
```



"Εργοστάσια" παραγωγής αντικειμένων (Object Factories) στο πρότυπο ES5

- Είναι συναρτήσεις που **επιστρέφουν κυριολεκτικές αναπαραστάσεις** αντικειμένων
- Στο παράδειγμα η συνάρτηση `createPerson` λαμβάνει δύο ορίσματα `name` και `age` και επιστρέφει ένα αντικείμενο με τα χαρακτηριστικά `name` και `age` και τη μέθοδο `greet`.
- Όταν κληθεί με τα ορίσματα `John` και `30` επιστρέφει ένα αντικείμενο που ανατίθεται στη σταθερά `person`.

```
function createPerson(name, age) {  
  return {  
    name: name,  
    age: age,  
    greet: function () {  
      console.log(`Hello, my name is ${this.name}.`);  
    },  
  };  
}  
  
const person = createPerson("John", 30);  
person.greet(); // Τυπώνει "Hello, my name is John."
```

- Τελικά η μέθοδος `greet` επιδρά στο αντικείμενο `person` και τυπώνει ένα μήνυμα που χρησιμοποιεί το χαρακτηριστικό `name`.



Κατασκευάστριες συναρτήσεις (Constructor Functions) αντικειμένων στο πρότυπο ES5

- Είναι συναρτήσεις που χρησιμοποιούνται με τον τελεστή `new` και δημιουργούν ένα αντικείμενο με συγκεκριμένο **αποτύπωμα**.
- Στο παράδειγμα η συνάρτηση `constructor` `Person` λαμβάνει δύο ορίσματα `name` και `age` και επιστρέφει ένα αντικείμενο με τα χαρακτηριστικά `name` και `age` και τη μέθοδο `greet`.
- Η κλήση με το `new` και τα ορίσματα `John` και `30` επιστρέφει ένα αντικείμενο που ανατίθεται στη σταθερά `person`.

```
function Person(name, age) {  
  this.name = name;  
  this.age = age;  
  this.greet = function () {  
    console.log(`Hello, my name is ${this.name}.`);  
  };  
}  
  
const person = new Person("John", 30);  
person.greet(); // Τυπώνει "Hello, my name is John."
```

- Τελικά η μέθοδος `greet` επιδρά στο αντικείμενο `person` και τυπώνει ένα μήνυμα που χρησιμοποιεί το χαρακτηριστικό `name`.



Χρήση του `prototype` αντί της κυριολεκτικής αναπαράστασης

- Η χρήση του `Person.prototype.greet` προσθέτει τη μέθοδο στο πρωτότυπο αντικείμενο της `Person`. *Έτσι η μέθοδος είναι κοινή για όλα τα στιγμιότυπα του αντικειμένου*
- Το `this.greet = function() {...}` δημιουργεί αντίγραφο της μεθόδου `greet` σε κάθε στιγμιότυπο της `Person`
- Πρακτικά το αποτέλεσμα είναι το ίδιο, όμως η χρήση του `prototype` είναι πιο αποδοτική

```
function Person(name) {  
  this.name = name;  
}  
  
Person.prototype.greet = function () {  
  console.log(`Hello, my name is ${this.name}`);  
};  
  
let person1 = new Person("John");  
let person2 = new Person("Jane");  
person1.greet(); // Τυπώνει "Hello, my name is John"  
person2.greet(); // Τυπώνει "Hello, my name is Jane"  
  
function Person(name) {  
  this.name = name;  
  this.greet = function () {  
    console.log(`Hello, my name is ${this.name}`);  
  };  
}  
  
person1 = new Person("John");  
person2 = new Person("Jane");  
person1.greet(); // Τυπώνει "Hello, my name is John"  
person2.greet(); // Τυπώνει "Hello, my name is Jane"
```




Κατασκευάστριες συναρτήσεις και κληρονομικότητα μέσω του πρωτότυπου (prototypical inheritance)

- Οι κατασκευάστριες συναρτήσεις μπορούν να δημιουργήσουν αντικείμενα με κοινές μεθόδους αν αυτές οριστούν στο πρωτότυπο της κατασκευάστριας συνάρτησης
- Στο παράδειγμα τα αντικείμενα που παράγει η `Person` **κληρονομούν** τη μέθοδο `fullName` από το πρωτότυπο αντικείμενο

Η μέθοδος `fullName` βρίσκεται μόνο μια φορά στο πρωτότυπο και δεν δημιουργείται εκ νέου σε κάθε στιγμιότυπο αντικειμένου. Αποτέλεσμα είναι ο **αποδοτικότερος χειρισμός της μνήμης**

```
function Person(firstName, lastName) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
}  
  
Person.prototype.fullName = function () {  
  return this.firstName + " " + this.lastName;  
};  
  
const john = new Person("John", "Doe");  
const jane = new Person("Jane", "Smith");  
  
console.log(john.fullName()); // Τυπώνει "John Doe"  
console.log(jane.fullName()); // Τυπώνει "Jane Smith"
```



Άσκηση στα αντικείμενα στο πρότυπο ES5

Δημιουργήστε ένα αντικείμενο και με τους τρεις διαθέσιμους τρόπους του προτύπου ES5:

- Δημιουργήστε μια **constructor function** `Product` που λαμβάνει τις παραμέτρους `name` , `price` και `description` και δημιουργεί ένα αντικείμενο με αυτά τα χαρακτηριστικά. Το αντικείμενο να έχει και μια μέθοδο `displayInfo` που τυπώνει στην οθόνη τις τιμές των χαρακτηριστικών του αντικειμένου.
- Δημιουργήστε ένα **object literal** `productObj` που αναπαριστά ένα αντικείμενο με το όνομα `iPhone` , τιμή `1200€` και περιγραφή `Apple's smartphone` . Το αντικείμενο να έχει και μια μέθοδο `displayInfo` που τυπώνει στην οθόνη το όνομα, την τιμή και την περιγραφή του.
- Δημιουργήστε ένα **object factory** που επιστρέφει object literals της προηγούμενης μορφής.



Κλάσεις στο πρότυπο ES6

Οι **κλάσεις** στο πρότυπο ES6 είναι ένας πιο συνοπτικός και εκφραστικός τρόπος για να ορίζουμε αντικείμενα. Επιτρέπουν πιο παραδοσιακή σύνταξη OOP και επιτρέπουν καλύτερη οργάνωση του κώδικα την κληρονομικότητα **χωρίς την ανάγκη πρωτοτύπων**

```
// ES5

function PersonES5(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

PersonES5.prototype.fullName = function () {
  return this.firstName + " " + this.lastName;
};
```

```
// ES6

class PersonES6 {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  fullName() {
    return this.firstName + " " + this.lastName;
  }
}
```



Κλάσεις στο πρότυπο ES6

- Χρησιμοποιούμε τη δεσμευμένη λέξη `class` για να ορίσουμε την κλάση και τη μέθοδο `constructor` για να ορίσουμε τα χαρακτηριστικά του αντικειμένου.
- Η μέθοδος `greet` ορίζεται στο σώμα της κλάσης χωρίς τη χρήση του `this`.
- Όταν κληθεί το όνομα της κλάσης με τον τελεστή `new` τότε εκτελείται αυτόματα η μέθοδος `constructor` και αρχικοποιεί το αντικείμενο ανάλογα με τις παραμέτρους.

```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  greet() {  
    console.log(`Hello, my name is ${this.name}.`);  
  }  
}  
  
const person = new Person("John", 30);  
person.greet(); // Τυπώνει "Hello, my name is John."
```

- Η μέθοδος `greet` ορίζεται μόνο μια φορά στην κλάση και όχι σε κάθε στιγμιότυπο αντικειμένου, όπως στο πρότυπο ES5 με τη χρήση του `prototype`



Ιδιωτικά χαρακτηριστικά στο πρότυπο ES6

- Στο πρότυπο ES6 δεν υπάρχει ακόμη η λέξη κλειδί `private` (υπάρχει πρόταση για την εισαγωγή της στο πρότυπο)
- Όμως υπάρχει η σύνταξη με το **πρόθεμα** `#` **στο όνομα του χαρακτηριστικού εκτός του constructor**
- Χαρακτηριστικά και μέθοδοι που έχουν το πρόθεμα `#` δεν είναι ορατά εκτός της κλάσης, στο παράδειγμα έχουμε δύο ιδιωτικά χαρακτηριστικά `#firstName` και `#lastName`

```
class Person {
  #firstName;
  #lastName;

  constructor(firstName, lastName) {
    this.#firstName = firstName;
    this.#lastName = lastName;
  }

  fullName() {
    return this.#firstName + " " + this.#lastName;
  }
}

const john = new Person("John", "Doe");
console.log(john.fullName()); // Τυπώνει "John Doe"
console.log(john.#firstName);
// SyntaxError: Private field '#firstName' must be declared in an enclosing class
```

- Η μέθοδος `fullName` ενώνει σαν αλφαριθμητικά τα ιδιωτικά χαρακτηριστικά
- Όμως η πρόσβαση εκτός της κλάσης στο `#firstName` δίνει μήνυμα λάθους



Getters και setters στο πρότυπο ES6

- Δηλώνουμε δύο ιδιωτικά χαρακτηριστικά `#firstName` και `#lastName`
- Το `fullName` είναι χαρακτηριστικό που υπολογίζεται από τα ιδιωτικά χαρακτηριστικά
- Η μέθοδος του getter χρησιμοποιεί το πρόθεμα `get`
- Η μέθοδος του setter χρησιμοποιεί το πρόθεμα `set` και ανάθεση με αποδιάρθρωση (`const [firstName, lastName]...` θα αναφερθούμε αργότερα)

```
class Person {  
  #firstName;  
  #lastName;  
  
  constructor(firstName, lastName) {  
    this.#firstName = firstName;  
    this.#lastName = lastName;  
  }  
  
  get fullName() {  
    return this.#firstName + " " + this.#lastName;  
  }  
  
  set fullName(name) {  
    const [firstName, lastName] = name.split(" ");  
    this.#firstName = firstName;  
    this.#lastName = lastName;  
  }  
}  
  
const john = new Person("John", "Doe");  
console.log(john.fullName); // Τυπώνει "John Doe"  
  
john.fullName = "Jane Smith";  
console.log(john.fullName); // Τυπώνει "Jane Smith"
```



Στατικές μέθοδοι στο πρότυπο ES6

- Μια στατική μέθοδος ανήκει στην κλάση και μπορεί να κληθεί μέσω της κλάσης χωρίς την ανάγκη να δημιουργηθεί στιγμιότυπο
- Το ίδιο συμβαίνει και με τα στατικά χαρακτηριστικά
- Οι στατικές μέθοδοι μπορεί να επιστρέφουν στιγμιότυπα των κλάσεων που ανήκουν και είναι ένας εναλλακτικός τρόπος δημιουργίας αντικειμένων χωρίς τη χρήση του constructor.

```
class User {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  static createUser(userData) {  
    const { name, age } = userData;  
    if (!name || !age) {  
      throw new Error("Name and age are required.");  
    }  
    return new User(name, age);  
  }  
}  
  
const userData = { name: "John Doe", age: 30 };  
const user = User.createUser(userData);  
console.log(user); // Τυπώνει User { name: "John Doe", age: 30 }
```




Κληρονομικότητα στο πρότυπο ES6

- Χρήση του `extends` στον ορισμό της κλάσης `Employee` για να κληρονομήσει την κλάση `Person`.
- Η εντολή `super` στην `Employee` καλεί τον `constructor` της `Person` και περνά τις παραμέτρους `name` και `age`.
- Η παράμετρος `jobTitle` περνά στα αντικείμενα της `Employee`.
- Η μέθοδος `introduce` χρησιμοποιεί χαρακτηριστικά και από τις δύο κλάσεις.

```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  greet() {  
    console.log(`Hello, my name is ${this.name}.`);  
  }  
}  
  
class Employee extends Person {  
  constructor(name, age, jobTitle) {  
    super(name, age);  
    this.jobTitle = jobTitle;  
  }  
  
  introduce() {  
    console.log(  
      `Hello, my name is ${this.name} and I work as a ${this.jobTitle}.`  
    );  
  }  
}  
  
const employee = new Employee("John", 30, "Software Engineer");  
employee.greet();  
// Τυπώνει "Hello, my name is John."  
employee.introduce();  
// Τυπώνει "Hello, my name is John and I work as a Software Engineer."
```



(Fat) Arrow Functions

- Ορίζονται στο πρότυπο ES6 και είναι ένας τρίτος τρόπος ορισμού συναρτήσεων εκτός των function declarations η των function expressions.
- Παρέχουν πιο συντομική σύνταξη, ειδικά όταν το σώμα της συνάρτησης έχει μόνο μια έκφραση.
- Χρησιμοποιούμε τον τελεστή `=>` ανάμεσα στις παραμέτρους και το σώμα της συνάρτησης.
- Επιστρέφουν τιμή χωρίς `return`

```
// Function declaration

function sum(a, b) {
  return a + b;
}

// Function expression

var sum = function (a, b) {
  return a + b;
};

// Arrow function

const sum = (a, b) => a + b;
```



Arrow Functions και `this`

- Οι arrow functions **δεν έχουν το δικό τους `this`** αλλά κληρονομούν την τιμή του από το πλαίσιο του κώδικα που τις περιέχει
- Με τη χρήση arrow function στη μέθοδο `greet` εξασφαλίζουμε πως η συνάρτηση δεσμεύει την τιμή του `this` από το αντικείμενο `Person`
- Έτσι έχει πρόσβαση στο χαρακτηριστικό `name` ακόμη κι αν κληθεί έξω από την εμβέλεια της κλάσης.

```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  greet = () => {  
    console.log(`Hello, my name is ${this.name}.`);  
  };  
}  
  
const person = new Person("John", 30);  
person.greet(); // Τυπώνει "Hello, my name is John."  
  
const greet = person.greet;  
greet(); // Τυπώνει "Hello, my name is John."
```



Χωρίς arrow function

- Η μέθοδος `greet` είναι regular function και όπως πριν δηλώνουμε τη μεταβλητή `greet` σαν άλλο όνομα πρόσβασης στη μέθοδο `greet` της κλάσης `Person`.
- Η κλήση `greet()` έξω από την εμβέλεια της κλάσης `Person` δεσμεύει το global `this` που δεν έχει ορισμένο το `name`.
- Χρειάζεται να δεσμευτεί το `this` της `Person` στην `greet` για να μην υπάρχει αυτό το πρόβλημα.

```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  greet() {  
    console.log(`Hello, my name is ${this.name}.`);  
  }  
}  
  
const person = new Person("John", 30);  
person.greet(); // Τυπώνει "Hello, my name is John."  
  
const greet = person.greet;  
greet(); // TypeError: Cannot read properties of undefined (reading 'name')
```

```
constructor(name, age) {  
  this.name = name;  
  this.age = age;  
  this.greet = this.greet.bind(this)  
}
```



Arrow functions σαν callbacks

- Η χρήση τους σαν callbacks σε higher order functions (συναρτήσεις που δέχονται συναρτήσεις σαν παραμέτρους) οδηγεί στην ανάπτυξη πιο περιεκτικού και ευανάγνωστου κώδικα.
- Χωρίς arrow function:

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter(function (num) {
  return num % 2 === 0;
});
console.log(evenNumbers); // Τυπώνει [2, 4]
```

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter((num) => num % 2 === 0);
console.log(evenNumbers); // Τυπώνει [2, 4]
```

```
const fruits = ["apple", "banana", "cherry"];
const fruitLengths = fruits.map((fruit) => fruit.length);
console.log(fruitLengths); // Τυπώνει [5, 6, 6]
```

```
const employees = [
  { name: "Alice", age: 25, salary: 50000 },
  { name: "Bob", age: 30, salary: 60000 },
  { name: "Charlie", age: 35, salary: 70000 },
];

const sortBySalary = (a, b) => b.salary - a.salary;
const sortedBySalary = employees.sort(sortBySalary);
console.log(sortedBySalary);
```



Ασκήσεις στις Arrow Functions

Απαντήστε όπως στην άσκηση 1 με μια γραμμή που περιέχει μια μεταβλητή που της ανατίθεται η συνάρτηση. Ελέγξτε το αποτέλεσμα με το `console.log`.

1. Μια συνάρτηση που επιστρέφει το άθροισμα ενός πίνακα με αριθμούς

```
const sum = (nums) => nums.reduce((acc, num) => acc + num, 0);
```

2. Μια συνάρτηση που επιστρέφει τις συμβολοσειρές ενός πίνακα σε αντίστροφη σειρά
3. Μια συνάρτηση που επιστρέφει αντικείμενα μόνο με το χαρακτηριστικό name από ένα πίνακα που περιέχει αντικείμενα με χαρακτηριστικά name και age
4. Μια συνάρτηση που επιστρέφει τους μονούς αριθμούς ενός πίνακα με αριθμούς
5. Μια συνάρτηση που επιστρέφει το άθροισμα όλων των ηλικιών των αντικειμένων ενός πίνακα που περιέχει αντικείμενα με χαρακτηριστικά, όνομα και ηλικία



Ασκήσεις στις Arrow Functions

6. Μια συνάρτηση που επιστρέφει τα αντικείμενα με ηλικία μεγαλύτερη από 30 από ένα πίνακα με αντικείμενα με χαρακτηριστικά όνομα και ηλικία.
7. Μια συνάρτηση που δέχεται τα χαρακτηριστικά, μικρό και μεγάλο όνομα και επιστρέφει το ονοματεπώνυμο
8. Μια συνάρτηση που επιστρέφει το μεγαλύτερο αριθμό από ένα πίνακα αριθμών
9. *Μια συνάρτηση που επιστρέφει το μέσο όρο των ηλικιών των αντικειμένων ενός πίνακα με αντικείμενα με χαρακτηριστικά όνομα και ηλικία*
10. *Μια συνάρτηση που επιστρέφει μόνο τα σύμφωνα από μια συμβολοσειρά*



Template literals

Χρησιμοποιούμε τα backticks ``` και το `${x}` σαν πλαίσιο για τη μεταβλητή `x` για να παρεμβάλουμε μεταβλητές ή εκφράσεις σε συμβολοσειρές.

```
// ES6

const name = "John";
const age = 30;

console.log(`Hello, my name is ${name} and I am ${age} years old.`);

const price = 19.99;
const discount = 0.1;

console.log(`The price of the product is ${price}€,
  but with a ${discount * 100}% discount,
  the price is ${price - price * discount}€.`);
```

```
// ES5

var name = "John";
var age = 30;

console.log("Hello, my name is " + name + " and I am " + age + " years old.");

var price = 19.99;
var discount = 0.1;

var discountedPrice = price - price * discount;
console.log("The price of the product is $" + price);
console.log(", but with a " + discount * 100 + "% discount,");
console.log("the price is $" + discountedPrice + ".");
```



Εξ ορισμού παράμετροι συναρτήσεων

Στο πρότυπο ES6 μπορούν να οριστούν εξ ορισμού τιμές στις παραμέτρους και δεν υπάρχει η ανάγκη της σύγκρισης με το `undefined` ή της χρήσης του λογικού `or` `||`

```
// ES6

function sayHello(name = "World") {
  console.log(`Hello, ${name}!`);
}

sayHello(); // Τυπώνει "Hello, World!"
sayHello("John"); // Τυπώνει "Hello, John!"
```

```
// ES5

function sayHello(name) {
  name = name || "World";
  // ή αλλιώς
  // if (name === 'undefined')
  //   { name = "World";}
  console.log("Hello, " + name + "!");
}

sayHello(); // Τυπώνει "Hello, World!"
sayHello("John"); // Τυπώνει "Hello, John!"
```



Απροσδιόριστος αριθμός παραμέτρων

Στο πρότυπο ES6 δεν χρειάζεται η χρήση του `arguments` για το χειρισμό απροσδιόριστου αριθμού παραμέτρων:

```
// ES6

function sum(...args) {
  return args.reduce((acc, val) => acc + val, 0);
}

const result = sum(1, 2, 3, 4);
console.log(result); // Τυπώνει 10
```

```
// ES5

function sum() {
  var total = 0;
  for (var i = 0; i < arguments.length; i++) {
    total += arguments[i];
  }
  return total;
}

var result = sum(1, 2, 3, 4);
console.log(result); // Τυπώνει 10
```



Ομαδοποίηση παραμέτρων

Στο πρότυπο ES6 μπορούμε να ομαδοποιήσουμε τις παραμέτρους από μια θέση και μετά

```
function join(separator, ...args) {  
  return args.slice(1).join(separator);  
}  
  
const result = join("-", "a", "b", "c", "d");  
console.log(result); // Τυπώνει "b-c-d"
```



Τελεστής εξάπλωσης (spread)

Ο τελεστής `...` εξαπλώνει το όρισμά του στα συστατικά του και διευκολύνει τη σύνθεση περισσότερο πολύπλοκων δομών (λειτουργεί και με τα strings):

```
// ES6

const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const combined = [...arr1, ...arr2];

console.log(combined); // Τυπώνει [1, 2, 3, 4, 5, 6]

const obj1 = { x: 1, y: 2 };
const obj2 = { z: 3 };
const combined = { ...obj1, ...obj2 };

console.log(combined); // Τυπώνει { x: 1, y: 2, z: 3 }
```

```
// ES5

var arr1 = [1, 2, 3];
var arr2 = [4, 5, 6];

var combined = arr1.concat(arr2);
console.log(combined); // Τυπώνει [1, 2, 3, 4, 5, 6]

var obj1 = { x: 1, y: 2 };
var obj2 = { z: 3 };
var combined = {};
for (var prop in obj1) {
    combined[prop] = obj1[prop];
}
for (var prop in obj2) {
    combined[prop] = obj2[prop];
}

console.log(combined); // Τυπώνει { x: 1, y: 2, z: 3 }
```



Value unpacking με destructuring assignment

Το πρότυπο ES6 δίνει τη δυνατότητα εξαγωγής τιμών από πίνακες και αντικείμενα με τη χρήση της ανάθεσης με αποδιάρθρωση (destructuring assignment)

// ES6

```
const arr = [1, 2, 3];  
const [a, b, c] = arr;  
console.log(a, b, c); // Τυπώνει 1 2 3  
  
const obj = { x: 1, y: 2, z: 3 };  
const { x, y, z } = obj;  
console.log(x, y, z); // Τυπώνει 1 2 3
```

// ES5

```
var arr = [1, 2, 3];  
var a = arr[0];  
var b = arr[1];  
var c = arr[2];  
console.log(a, b, c); // Τυπώνει 1 2 3  
  
var obj = { x: 1, y: 2, z: 3 };  
var x = obj.x;  
var y = obj.y;  
var z = obj.z;  
console.log(x, y, z); // Τυπώνει 1 2 3
```



Εναλλαγή τιμών με ανάθεση αποδιάρθρωσης

Η εναλλαγή τιμών δύο μεταβλητών χρειάζεται παραδοσιακά και μια τρίτη μεταβλητή, όχι όμως στο πρότυπο ES6:

```
// ES6
```

```
let first = "world";  
let second = "hello";  
  
[first, second] = [second, first];  
  
console.log(`${first} ${second}`);  
// 'hello world'
```

```
// ES5
```

```
var first = "world";  
var second = "hello";  
var temp = first;  
  
first = second;  
second = temp;  
  
console.log(first + " " + second);  
// 'hello world'
```




Aliases μεθόδων με destructuring assignment

Στο πρότυπο ES6 μπορούμε να έχουμε πολλαπλά ονόματα για τις μεθόδους ενός αντικειμένου κάνοντας χρήση του destructuring assignment:

```
// ES6

const obj = {
  sayHello() {
    console.log("Hello!");
  },
};

// Method alias using destructuring assignment
const { sayHello: greet } = obj;

greet(); // Τυπώνει "Hello!"
```

```
// ES6

class MyClass {
  sayHello(message) {
    console.log(`Hello, ${message}`);
  }
  sayBye(message) {
    console.log(`Bye, ${message}`);
  }
}

let myClass = new MyClass();
let { sayHello: hello, sayBye: bye } = myClass;

hello("how are you?"); // Hello, how are you?
bye("see you soon"); // Bye, see you soon
```



Εξ ορισμού τιμές με αποδιάρθρωση αντικειμένου

Στο πρότυπο ES6 μπορούμε να έχουμε εξ ορισμού τιμές κατά την αποδιάρθρωση των χαρακτηριστικών ενός αντικειμένου:

```
const obj = {  
  name: "John",  
  age: 30,  
};  
  
// Destructuring with default value  
const { name, occupation = "unknown" } = obj;  
  
console.log(name); // Τυπώνει "John"  
console.log(occupation); // Τυπώνει "unknown"
```



Ενότητες (modules) κώδικα

Το πρότυπο ES6 υποστηρίζει την εισαγωγή και εξαγωγή τιμών από και προς ενότητες κώδικα χωρίς να επηρεάζεται το global namespace

```
// ES6
// module lib/logger.js

export function log(message) {
  console.log(message);
}
export var defaultErrorMessage = "Aw, Snap!";

// myApp.js
import * as logger from "lib/logger";
logger.log(logger.defaultErrorMessage);

// anotherApp.js
import { log, defaultErrorMessage } from "lib/logger";
log(defaultErrorMessage);
```

```
// ES5
// lib/logger.js

LoggerLib = {};
LoggerLib.log = function (message) {
  console.log(message);
};
LoggerLib.defaultErrorMessage = "Aw, Snap!";

// myApp.js
var logger = LoggerLib;
logger.log(logger.defaultErrorMessage);

// anotherApp.js
var log = LoggerLib.log;
var defaultErrorMessage = LoggerLib.defaultErrorMessage;
log(defaultErrorMessage);
```



Πολλαπλές και εξ ορισμού εξαγωγές από ενότητες κώδικα

Οι πολλαπλές εξαγωγές (wildcard exports) είναι ιδιαίτερα χρήσιμες όταν δημιουργούμε μια σύνθετη ενότητα που επαναεξάγει κώδικα από άλλες ενότητες

```
// ES6

// lib/complex-module.js
export * from "lib/logger";
export * from "lib/http";
export * from "lib/utils";

// app.js
import { logger, httpClient, stringUtils } from "lib/complex-module";
logger.log("hello from logger");
```

Εξ ορισμού εξαγωγές γίνονται με χρήση του
export default

```
// ES6

// lib/logger.js
export default (message) => console.log(message);

// app.js
import output from "lib/logger";
output("hello world");
```



Ασκήσεις στις κλάσεις

1. Γράψτε μια κλάση `Rectangle` με δύο χαρακτηριστικά `width` και `height`. Η κλάση πρέπει να έχει μια μέθοδο `area` που επιστρέφει το εμβαδόν του παραλληλογράμμου. Παράδειγμα εξόδου:

```
const rect = new Rectangle(4, 5);  
console.log(rect.area()); // Τυπώνει 20
```

2. Γράψτε μια κλάση `Square` που επεκτείνει τη `Rectangle`. Η `Square` πρέπει να έχει μέθοδο `constructor` που λαμβάνει ένα όρισμα `side` και το αναθέτει και στα δύο χαρακτηριστικά `width` και `height` και να κληρονομεί τη μέθοδο `area`. Παράδειγμα εξόδου:

```
const square = new Square(5);  
console.log(square.area()); // Τυπώνει 25
```



Ασκήσεις στις κλάσεις

3. Μετατρέψτε τα χαρακτηριστικά `width` και `height` της `Rectangle` σε `private`. Προσθέστε στην `Rectangle` `getters` και `setters` για τα `private` χαρακτηριστικά `width` και `height`.

Παράδειγμα εξόδου:

```
const rect = new Rectangle(4, 5);
console.log(rect.width); // Τυπώνει undefined
console.log(rect.height); // Τυπώνει undefined
console.log(rect.getWidth()); // Τυπώνει 4
console.log(rect.getHeight()); // Τυπώνει 5
rect.setWidth(6);
console.log(rect.getWidth()); // Τυπώνει 6
console.log(rect.area()); // Τυπώνει 30
```



Ασκήσεις στις κλάσεις

4. Προσθέστε μια στατική μέθοδο στην κλάση `Rectangle` που επιστρέφει την περίμετρο του παραλληλογράμμου. Παράδειγμα εξόδου:

```
console.log(Rectangle.perimeter(4, 5)); // Τυπώνει 18
```

5. Γράψτε μια κλάση `Circle` με ένα χαρακτηριστικό `radius` και μια μέθοδο `area` που επιστρέφει το εμβαδόν του κύκλου με αυτή την ακτίνα. Η κλάση να ανατεθεί σε μια μεταβλητή `circle`. Παράδειγμα εξόδου:

```
const circle = new Circle(5);  
console.log(circle.area()); // Τυπώνει 78.53981633974483
```




Ασκήσεις στην αποδιάρθρωση

1. Γράψτε μια συνάρτηση που λαμβάνει ένα αντικείμενο χρήστη στην είσοδο και επιστρέφει το όνομα και την διεύθυνση email του. Το αντικείμενο μπορεί να έχει και άλλα χαρακτηριστικά που δεν θα επιστρέφονται.

Παράδειγμα input:

```
const user = {  
  name: "John Doe",  
  age: 30,  
  email: "john@example.com",  
  phone: "555-555-5555",  
};
```

Παράδειγμα output:

```
{  
  name: "John Doe",  
  email: "john@example.com"  
}
```



Ασκήσεις στην αποδιάρθρωση

2. Γράψτε μια συνάρτηση που λαμβάνει ένα πίνακα αριθμών στην είσοδο και επιστρέφει το άθροισμα των δύο πρώτων αριθμών.

Παράδειγμα input:

```
const nums = [1, 2, 3, 4, 5];
```

Παράδειγμα output:

3



Ασκήσεις στην αποδιάθρωση

3. Γράψτε μια συνάρτηση που λαμβάνει ένα αντικείμενο χρήστη στην είσοδο και επιστρέφει το όνομα και την ηλικία του. Αν το αντικείμενο δεν έχει χαρακτηριστικό ηλικίας τότε θα επιστρέφει την εξ ορισμού τιμή 0.

Παράδειγμα input:

```
const user1 = {  
  name: "Alice",  
  age: 25,  
};  
  
const user2 = {  
  name: "Bob",  
};
```

Παράδειγμα output:

```
{  
  name: "Alice",  
  age: 25  
}  
  
{  
  name: "Bob",  
  age: 0  
}
```



Ασκήσεις στην αποδιάρθρωση

4. Γράψτε μια συνάρτηση που λαμβάνει στην είσοδο ένα πίνακα με αριθμούς και επιστρέφει το άθροισμα των δύο πρώτων και τους υπόλοιπους αριθμούς του πίνακα σαν ένα πίνακα.

Παράδειγμα input:

```
const nums = [1, 2, 3, 4, 5];
```

Παράδειγμα output:

```
{  
  sum: 3,  
  rest: [3, 4, 5]  
}
```



Ασκήσεις στην αποδιάθρωση

5. Γράψτε μια συνάρτηση που λαμβάνει στην είσοδο ένα αντικείμενο `person` που ενσωματώνει ένα άλλο αντικείμενο `address` και επιστρέφει τό όνομα του προσώπου και την πόλη διαμονής του.

Παράδειγμα input:

```
const person = {  
  name: "Jane Doe",  
  age: 35,  
  address: {  
    street: "123 Main St",  
    city: "New York",  
    state: "NY",  
    zip: "10001",  
  },  
};
```

Παράδειγμα output:

```
{  
  name: "Jane Doe",  
  city: "New York"  
}
```



Ασκήσεις στην αποδιάρθρωση

6. Γράψτε μια συνάρτηση που λαμβάνει στην είσοδο ένα πίνακα αριθμών και επιστρέφει το άθροισμα των δύο πρώτων αριθμών του πίνακα. Αν ο πίνακας έχει λιγότερα από δύο στοιχεία τότε να χρησιμοποιηθεί η εξ ορισμού τιμή 0 για τα στοιχεία που δεν υπάρχουν.

Παράδειγμα input:

```
const nums1 = [1, 2];  
const nums2 = [1];  
const nums3 = [];
```

Παράδειγμα output:

```
3  
2  
0
```



Μεταπρογραμματισμός και αντανάκλαση

- Λέμε πως κάνουμε **μεταπρογραμματισμό (metaprogramming)** όταν προγραμματίζουμε ένα πρόγραμμα να ελέγχει ή και να αλλάζει τον εαυτό του και τη δομή του κατά τη διάρκεια της εκτέλεσής του
- Η **αντανάκλαση (reflection)** στη Javascript επιτρέπει τον μεταπρογραμματισμό, αρχικά με τη χρήση μεθόδων όπως τα `Object.keys` ή `Object.getOwnPropertyNames` ενώ με την εισαγωγή του **Reflect** στο πρότυπο ECMAScript 6 υπάρχει πλέον πρότυπο API για μεταπρογραμματισμό
- Όπως η αντανάκλασή μας στον καθρέπτη επιτρέπει να δούμε από τον εαυτό μας πράγματα που δεν μπορούμε να κοιτάξουμε (μάτια, χείλια, κτλ) και να επέμβουμε σε αυτά, έτσι και η αντανάκλαση στον κώδικα μας επιτρέπει να εξετάσουμε τα αντικείμενα που προκύπτουν κατά τη διάρκεια της εκτέλεσης και να επέμβουμε, ενδεχομένως τροποποιώντας τα



Reflect API

Όλες οι μέθοδοι είναι στατικές

Μέθοδος	Πληροφορίες
<code>Reflect.apply()</code>	call a function with specified arguments
<code>Reflect.construct()</code>	act like the <code>new</code> operator, but as a function. It is equivalent to calling <code>new target(...args)</code>
<code>Reflect.defineProperty()</code>	is similar to <code>Object.defineProperty()</code> , but return a Boolean value indicating whether or not the property was successfully defined on the object
<code>Reflect.deleteProperty()</code>	behave like the <code>delete</code> operator, but as a function. It's equivalent to calling the <code>delete objectName[propertyName]</code>
<code>Reflect.get()</code>	return the value of a property
<code>Reflect.getOwnPropertyDescriptor()</code>	is similar to <code>Object.getOwnPropertyDescriptor()</code> . It returns a property descriptor of a property if the property exists on the object, or <code>undefined</code> otherwise
<code>Reflect.getPrototypeOf()</code>	is the same as <code>Object.getPrototypeOf()</code>
<code>Reflect.has()</code>	work like the <code>in</code> operator, but as a function. It returns a boolean indicating whether an property (either owned or inherited) exists
<code>Reflect.isExtensible()</code>	is the same as <code>Object.isExtensible()</code>
<code>Reflect.ownKeys()</code>	return an array of the owned property keys (not inherited) of an object
<code>Reflect.preventExtensions()</code>	is similar to <code>Object.preventExtensions()</code> . It returns a Boolean.
<code>Reflect.set()</code>	assign a value to a property and return a Boolean value which is true if the property is set successfully
<code>Reflect.setPrototypeOf()</code>	set the prototype of an object



Χρήση του Reflect.construct

- Η κλήση επιτρέπει ένα νέο στιγμιότυπο του `target`, ή του `newTarget` αν οριστεί, αρχικοποιημένο από τον constructor του `target` με τα *array like* ορίσματα `args`

```
Reflect.construct(target, args [, newTarget])
```

- Είναι ανάλογο με την κλήση

```
new target(...args);
```

- Δίνει όμως τη δυνατότητα χρήσης και διαφορετικού target

```
class Person {  
  constructor(firstName, lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
  }  
  get fullName() {  
    return `${this.firstName} ${this.lastName}`;  
  }  
}  
  
let args = ["John", "Doe"];  
  
let john = Reflect.construct(Person, args);  
  
console.log(john instanceof Person);  
console.log(john.fullName); // John Doe
```



Reflect.construct με newTarget

- `Employee` είναι υποκλάση της `Person` και προσθέτει τα χαρακτηριστικά `title` και `salary`
- Η μεταβλητή `args` περιέχει τα ορίσματα του constructor της `Employee`
- Η μεταβλητή `newEmployee` χρησιμοποιεί το `Reflect.construct` για να αρχικοποιήσει την κλάση `Employee`, επιστρέφει όμως στιγμιότυπο της `Person`

```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
}  
  
class Employee extends Person {  
  constructor(name, age, title, salary) {  
    super(name, age);  
    this.title = title;  
    this.salary = salary;  
  }  
}  
  
const args = ["John Doe", 30, "Software Engineer", 100000];  
const newEmployee = Reflect.construct(Employee, args, Person);  
console.log(newEmployee);
```



Χρήση του Reflect.defineProperty

Λειτουργεί όπως το `Object.defineProperty` από το ES5, επιστρέφει όμως ένα Boolean σαν ένδειξη για το αν το χαρακτηριστικό ορίστηκε, αντί εξαίρεσης κατά την εκτέλεση

Στο παράδειγμα ορίζεται το χαρακτηριστικό `age` που

- μπορεί να αλλαχτεί η τιμή του (writable)
- μπορεί να αλλάξει ή να διαγραφεί σαν χαρακτηριστικό (configurable)
- δεν θα αναφέρεται στο `for...in` στο στιγμιότυπο του αντικειμένου
- έχει την τιμή `25`

```
let person = {
  name: "John Doe",
};

if (
  Reflect.defineProperty(person, "age", {
    writable: true,
    configurable: true,
    enumerable: false,
    value: 25,
  })
) {
  console.log(person.age);
} else {
  console.log("Cannot define the age property on the person object.");
}
```

Το τρίτο όρισμα είναι στιγμιότυπο της ενσωματωμένης από το ES5 διεπαφής

PropertyDescriptor



Η διεπαφή `PropertyDescriptor`

Είναι μια διεπαφή που περιγράφει τις ιδιότητες των αντικειμένων. Χρησιμοποιείται από διάφορες μεθόδους της γλώσσας που ορίζουν ή αλλάζουν τις ιδιότητες

- `value` είναι η τιμή της ιδιότητας, αν πρόκειται για χαρακτηριστικό δεδομένων
- `writable` είναι `boolean` ένδειξη για το αν μπορεί να αλλαχτεί η τιμή
- `enumerable` είναι `boolean` ένδειξη αν η ιδιότητα θα αναφέρεται στο βρόγχο `for...in` του αντικειμένου
- `configurable` είναι `boolean` ένδειξη αν η ιδιότητα μπορεί να διαγραφεί ή αλλάξει με τη χρήση του `Reflect.defineProperty`
- `get` είναι η συνάρτηση που καλείται όταν η ιδιότητα διαβάζεται
- `set` είναι η συνάρτηση που καλείται όταν η ιδιότητα αλλάζει

Διάφορες ιδιότητες είναι αμοιβαία αποκλειόμενες (π.χ. `value`, `writable=false`)



Παραδείγματα χρήσης του Reflect

1. Μια συνάρτηση που τυπώνει το όνομα και την τιμή μιας ιδιότητας ενός αντικειμένου

```
function logProperty(obj, propertyName) {  
  const value = Reflect.get(obj, propertyName);  
  console.log(`${propertyName}: ${value}`);  
}  
  
const obj = { name: "John", age: 30 };  
logProperty(obj, "age");
```

Τυπώνει

```
age: 30
```



Παραδείγματα χρήσης του Reflect

2. Μια συνάρτηση δέχεται ένα αντικείμενο και τυπώνει όλες τις ιδιότητές του με τις τιμές τους

```
function logAllProperties(obj) {  
  const properties = Reflect.ownKeys(obj);  
  properties.forEach((propertyName) => {  
    const value = Reflect.get(obj, propertyName);  
    console.log(`${propertyName}: ${value}`);  
  });  
}  
  
const person = { name: "John", age: 30, country: "Greece" };  
const car = { brand: "Toyota", model: "Yaris", year: 1998 };  
logAllProperties(person);  
logAllProperties(car);
```

Τυπώνει

```
name: John  
age: 30  
country: Greece  
brand: Toyota  
model: Yaris  
year: 1998
```



Παραδείγματα χρήσης του Reflect

3. Μια συνάρτηση που αντιγράφει τις ιδιότητες ενός αντικειμένου σε ένα άλλο

```
function copyProperties(source, target) {  
  const properties = Reflect.ownKeys(source);  
  properties.forEach((propertyName) => {  
    const descriptor = Reflect.getOwnPropertyDescriptor(source, propertyName);  
    Reflect.defineProperty(target, propertyName, descriptor);  
  });  
}  
  
const person = { name: "John", age: 30, country: "Greece" };  
const student = { school: "Mechanical Engineer" };  
copyProperties(person, student);  
logAllProperties(student);
```

Τυπώνει

```
school: Mechanical Engineer  
name: John  
age: 30  
country: Greece
```



Παραδείγματα χρήσης του Reflect

4. Μια συνάρτηση που περιβάλλει μια μέθοδο ενός αντικειμένου με μια συνάρτηση που τυπώνει το όνομα και τα ορίσματα της μεθόδου που καλείται πριν καλέσει τη μέθοδο με τα ορίσματα

```
function logMethodCalls(obj: any, methodName: string) {
    const originalMethod = Reflect.get(obj, methodName);
    Reflect.defineProperty(obj, methodName, {
        value: function (...args: any[]) {
            console.log(`${methodName}(${args.join(", ")} )`);
            return Reflect.apply(originalMethod, obj, args);
        },
    });
}

const obj = {
    name: "John",
    greet: function (greeting) {
        console.log(`${greeting}, my name is ${this.name}.`);
    },
};
logMethodCalls(obj, "greet");
obj.greet("Hello");
```

Τυπώνει

```
greet(Hello)
Hello, my name is John.
```




Παραδείγματα χρήσης του Reflect

5. Μια συνάρτηση που "παγώνει" ένα αντικείμενο μετατρέποντας όλες τις ιδιότητες του σε μη εγγράψιμες (non-writable) και μη διαμορφώσιμες (non-configurable)

```
function freezeObject(obj) {  
  const properties = Reflect.ownKeys(obj);  
  properties.forEach((propertyName) => {  
    const descriptor = Reflect.getOwnPropertyDescriptor(obj, propertyName);  
    if (descriptor && "value" in descriptor) {  
      descriptor.writable = false;  
      descriptor.configurable = false;  
      Reflect.defineProperty(obj, propertyName, descriptor);  
    }  
  });  
  Object.freeze(obj);  
}  
const obj = { name: "John", age: 30 };  
freezeObject(obj);  
obj.age = 60;  
console.log(obj);
```

Τυπώνει

```
{ name: 'John', age: 30 }
```