

Τεχνολογία Διαδικτύου

5. Javascript 2

Γρηγόρης Τζιάλλας

Καθηγητής

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Σχολή Θετικών Επιστημών

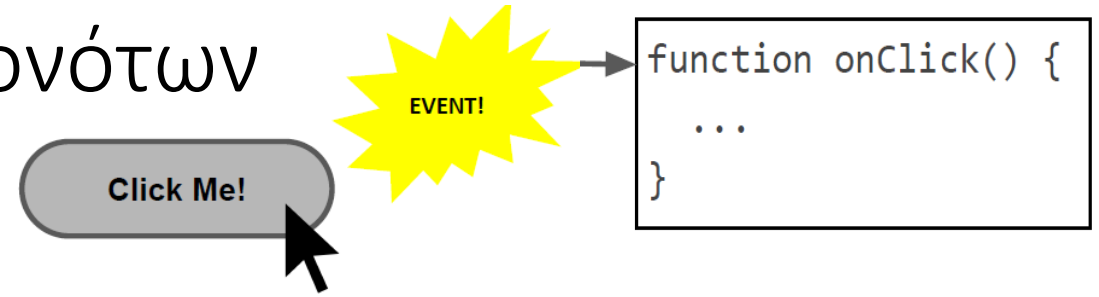
Πανεπιστήμιο Θεσσαλίας

Προγραμματισμός χειρισμού γεγονότων

Event-driven programming

Προγραμματισμός χειρισμού γεγονότων

Event-driven programming



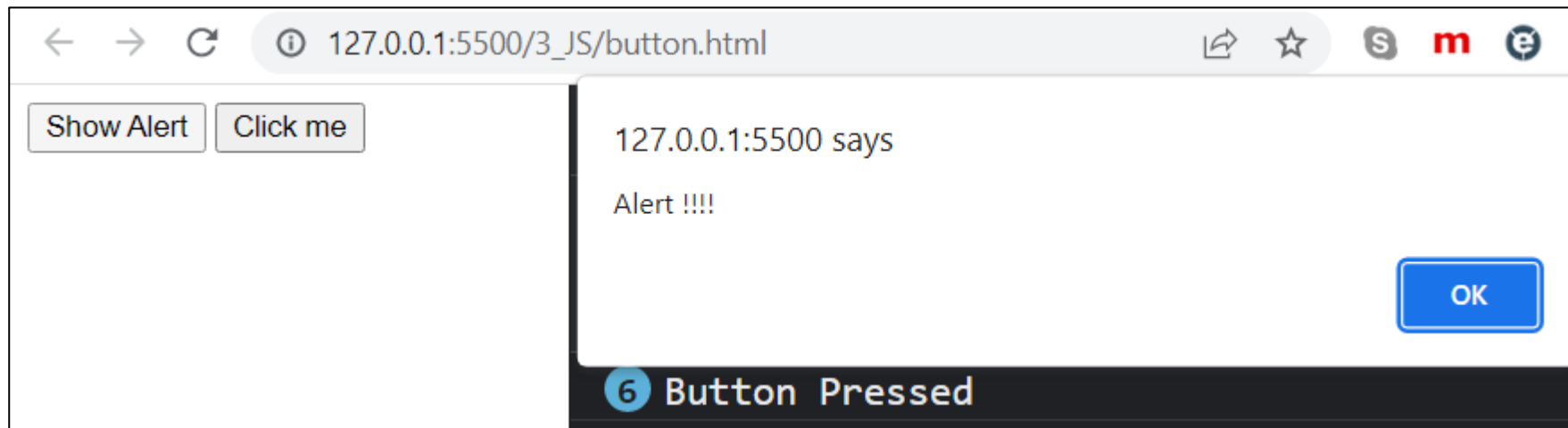
- Ο διαδικτυακός προγραμματισμός στηρίζεται κυρίως στην διαχείριση γεγονότων (event-driven)
- Ο κώδικας εκτελείται όταν συμβαίνει κάποιο γεγονός
- Ένα γεγονός προκαλείται από το πάτημα ενός πλήκτρου, την πληκτρολόγηση, την κίνηση του ποντικιού, από συσκευές εισόδου και γενικότερα με την αλληλεπίδραση με τον χρήστη και με άλλα εξωτερικά συστήματα.
- Η διαχείριση των γεγονότων γίνεται με την διασύνδεση τους με συναρτήσεις για τον χειρισμό τους (event handlers)

Παράδειγμα χειρισμού γεγονότων

```
function handleClick(){  
    console.log("Button Pressed");  
}
```

Inline χειρισμός συμβάντων

```
<body>  
  <button onclick="alert('Alert !!!!');">Show Alert</button>  
  <button onclick="handleClick()">Click me</button>  
</body>
```



DOM

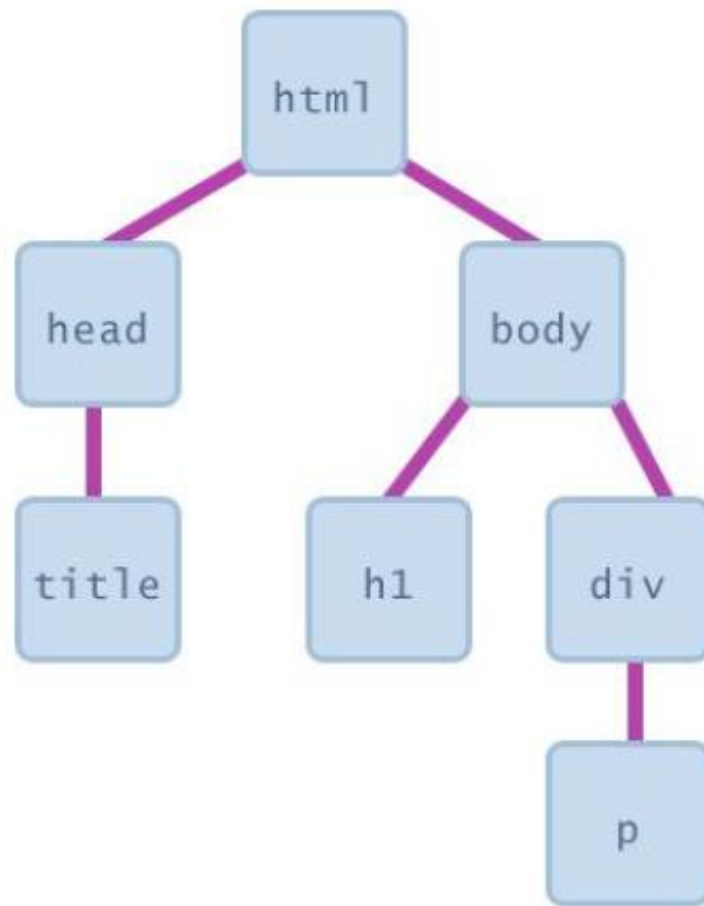
Document Object Model

DOM – Document Object Model

- Το DOM είναι ένα ιεραρχικό δένδρο κόμβων στο οποίο αντιστοιχούν τα διάφορα στοιχεία της ιστοσελίδας
- Ο κώδικας Javascript μπορεί να χρησιμοποιήσει το DOM για:
 - να εξετάσει την κατάσταση των στοιχείων της ιστοσελίδας και να διαβάσει τα περιεχόμενα τους (πχ. το κείμενο που εισήγαγε ο χρήστης)
 - να αλλάξει δυναμικά τις ιδιότητες των στοιχείων (πχ. να αλλάξει το περιεχόμενο, την μορφή και την συμπεριφορά τους)
 - να προσθέσει ή διαγράψει στοιχεία της ιστοσελίδας

Παράδειγμα DOM

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <h1></h1>
    <div>
      <p></p>
    </div>
  </body>
</html>
```



Τύποι κόμβων DOM

- Το DOM εκτός από κόμβους που αντιστοιχούν σε στοιχεία HTML (elements) περιλαμβάνει και άλλου τύπου κόμβους όπως:
 - text nodes που αντιστοιχούν στο κείμενο που ορίζουν τα στοιχεία html
 - HTML comments
 - κ.λπ.
- Ο τύπος του κάθε node αποθηκεύεται στην ιδιότητα [nodeType](#)

Βασικές ιδιότητες DOM elements (1)

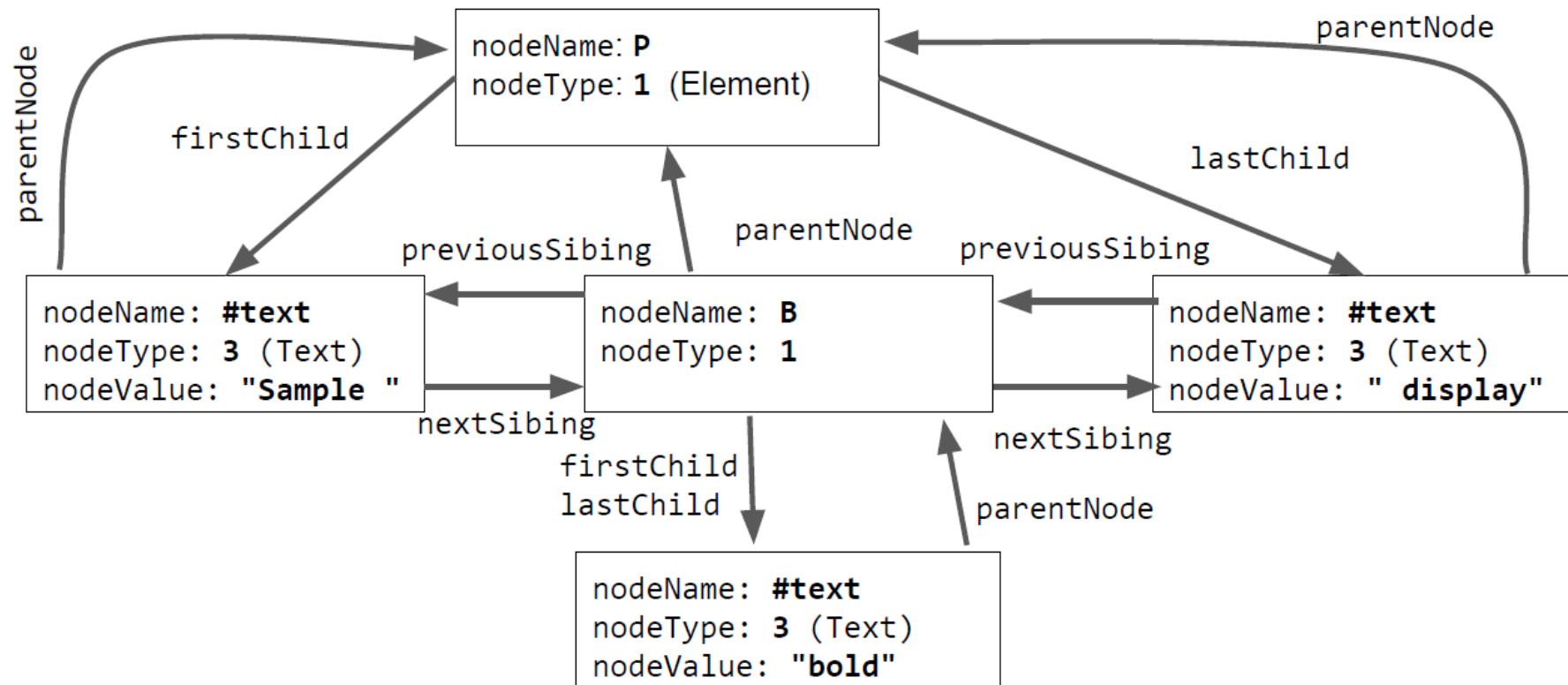
- `id`
Η ιδιότητα `id` του στοιχείου HTML
- `innerHTML`
Ο κώδικας HTML μεταξύ των ετικετών αρχής και τέλους του στοιχείου
- `textContent`
Το κείμενο που περιέχει το στοιχείο και τα περιεχόμενα του
- `classList`
Οι κλάσεις CSS του στοιχείου
- `parentNode`
Το γονικό στοιχείο που το περιέχει
- `childNodes`
Λίστα με τα στοιχεία που περιέχει

Βασικές ιδιότητες DOM elements (2)

- `outerHTML`: Παρόμοιο με το `innerHTML` με την διαφορά ότι επιστρέφει και την ετικέτα του στοιχείου
- `getAttribute()/setAttribute()`: Επιστρέφει η καθορίζει την ιδιότητα ενός στοιχείου
- `nextSibling`, `previousSibling`, `firstChild`, `lastChild` ..
- Μέσω της ιδιότητας `style` μπορούμε να αλλάξουμε την εμφάνιση ή να αποκρύψουμε ένα στοιχείο
 - `element.style.display = "none";`
 - `element.style.display = "";`

Σχέσεις κόμβων DOM

`<p>Sample bold display</p>`



Προσπέλαση στα στοιχεία του DOM

Η προσπέλαση γίνεται με επιλογείς όπως και στους κανόνες CSS

- `document.querySelector('css selector');`
 - Επιστρέφει το πρώτο στοιχείο που επιλέγει ο επιλογέας
- `document.querySelectorAll('css selector');`
 - Επιστρέφει όλα τα στοιχεία που επιλέγει ο επιλογέας

Παλαιότεροι τρόποι:

- `getElementById()`
- `getElementsByTagName()`
- `getElementsByClassName()`
- Αντιστοίχιση στοιχείων html σε αντικείμενα javascript:
 - `html` `window.document.documentElement`
 - `head` `window.document.head`
 - `Body` `window.document.body`

Κόμβοι DOM

```
function walkTree(root, level) {  
  if (root.nodeType === Node.TEXT_NODE) {  
    console.log(level + 'text:' + root.textContent);  
  } else {  
    console.log(level + root.nodeName);  
  }  
  for (const child of root.childNodes) {  
    walkTree(child, level + "  ");  
  }  
}  
walkTree(document.querySelector('html'), "");
```

Προσπέλαση όλων των κόμβων του DOM με αναδρομή

Πρόσβαση στις ιδιότητες των στοιχείων

- Οι ιδιότητες των στοιχείων html είναι προσβάσιμες μέσω των αντίστοιχων αντικειμένων του DOM

πχ.

HTML

```

```

JavaScript

```
const element = document.querySelector('img');  
element.src = 'bear.png';
```

Παράδειγμα αλλαγής ιδιοτήτων

```
function showImage() {  
    const anImage= document.querySelector("img");  
    anImage.width=200;  
    anImage.src="/images/logo-uth.png";  
    anImage.alt="CS Logo";  
}
```

```
<div>  
    <button onclick="showImage()">Show Image</button>  
</div>  
<div>  
    <img/>  
</div>
```

Show Image



Προσθήκη και διαγραφή κλάσεων

- Η ιδιότητα [classList](#) ενός node element διαθέτει τις μεθόδους add και remove για την προσθήκη νέων κλάσεων ή την διαγραφή τους.

```
function hideImage() {  
    const anImage= document.querySelector("img");  
    anImage.classList.add("hidden");  
}  
function showHidden() {  
    const allHidden = document.querySelectorAll(".hidden");  
    for (const anElement of allHidden){  
        anElement.classList.remove("hidden");  
    }  
}
```

```
.hidden{  
    display:none;  
}
```

```
<div>  
      
</div>  
<div class="hidden">Πανεπιστήμιο Θεσσαλίας</div>  
<div>  
    <button onclick="hideImage()">Hide Image</button>  
    <button onclick="showHidden()">Show hidden</button>  
</div>
```


Προσθήκη και διαγραφή στοιχείων του DOM

Πρόσθεση νέων στοιχείων

- Με τις μεθόδους:
 `document.createElement(tag string)`
 `parent.appendChild(element);`
 μπορούμε να προσθέσουμε δυναμικά στοιχεία στο DOM

Διαγραφή στοιχείων

- Με την μέθοδο
 `element.remove();`
 μπορούμε να αφαιρέσουμε ένα στοιχείο του DOM
- Εναλλακτικά με την ιδιότητα `innerHTML` μπορούμε να διαγράψουμε όλα τα περιεχόμενα ενός στοιχείου
 `element.innerHTML = '';`

Παράδειγμα πρόσθεσης και αφαίρεσης στοιχείων

```
function addElement() {  
    let aNewElement = document.createElement("div");  
    aNewElement.textContent="New Element";  
    let aParent = document.querySelector("#Container");  
    aParent.appendChild(aNewElement);  
}  
function removeElement() {  
    let anElement = document.querySelector("#Container div");  
    anElement.remove();  
}  
function removeAll() {  
    let aParent = document.querySelector("#Container");  
    aParent.innerHTML="";  
}
```

```
<div>  
    <button onclick="addElement()">Προσθήκη στοιχείου</button>  
    <button onclick="removeElement()">Διαγραφή στοιχείου</button>  
    <button onclick="removeAll()">Διαγραφή όλων</button>  
</div>  
<div id="Container"></div>
```

Δεδομένα στοιχείων html

- Μπορούμε να ορίσουμε δεδομένα σε ένα στοιχείο html τα οποία μπορούμε να διαβάσουμε από το DOM με την χρήση ιδιοτήτων που το όνομά τους ξεκινούν με data-


π.χ. `<div data-name= "value">`

- Η πρόσβαση από το DOM γίνεται με την ιδιότητα dataset

π.χ. `element.dataset.name //returns "value"`

Παράδειγμα δεδομένων στοιχείων html



currentTarget : ιδιότητα που επιστρέφει το στοιχείο που αποστέλλει το γεγονός



```
function handleClick(event) {  
    let anElement = event.currentTarget;  
    let clickCount = parseInt(anElement.dataset.clickcount) + 1;  
    console.log(`Element ${anElement.textContent} clicks ${clickCount}`);  
    if (clickCount >= 3){  
        anElement.remove();  
    }else{  
        anElement.dataset.clickcount = clickCount;  
    }  
}
```

Ορισμός ιδιότητας δεδομένου

Παράμετρος event γεγονότος



```
<h1>Click on elements 3 times to remove them</h1>  
<div data-clickCount="0" onclick="handleClick(event)">1</div>  
<div data-clickCount="0" onclick="handleClick(event)">2</div>  
<div data-clickCount="0" onclick="handleClick(event)">3</div>
```

Δυναμικός χειρισμός γεγονότων μέσω DOM

Η συσχέτιση γεγονότων με συναρτήσεις για το χειρισμό τους γίνεται με την συνάρτηση:

- `addEventListener(event name, function name);`
 - `event name` είναι το όνομα (string) του συμβάντος που θέλουμε να ακούσουμε πχ. `click`, `focus`, `blur`, `mouseover`,...
 - `function name` είναι το όνομα της συνάρτησης που θα χειρισθεί το συμβάν


Η διαγραφή ενός χειριστή γεγονότος μπορεί να γίνει με την συνάρτηση.

- `removeEventListener(event name, function name);`

Παράδειγμα addEventListener

```
function onClick() {  
    console.log("Button Clicked");  
}  
const buttonClick = document.querySelector("#ButtonClick");  
buttonClick.addEventListener('click', onClick);
```

Η χρήση της δήλωσης [defer](#) είναι απαραίτητη για να ολοκληρωθεί η φόρτωση των στοιχείων της ιστοσελίδας και να επιλεγούν με την μέθοδο `querySelector`



```
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>DOM</title>  
    <script src="dom.js" defer></script>  
</head>  
  
<body>  
    <button id="ButtonClick">Click Me</button>  
</body>
```

Αντικείμενα Javascript

Αντικείμενα Javascript

- Στη JavaScript, σχεδόν όλα είναι αντικείμενα.
 - Οι τύποι Date, Math, Array, Regular Expression, και Function είναι αντικείμενα
 - Οι μεταβλητές τύπου boolean, number, και οι συμβολοσειρές εάν δημιουργούνται με την λέξη κλειδί new είναι αντικείμενα
 - Όλες οι τιμές JavaScript, εκτός από τις πρωταρχικές (primitives), είναι αντικείμενα.
- Οι βασικοί τύποι δεδομένων (primitive data types) δεν έχουν ιδιότητες και μεθόδους
- Οι τιμές των βασικών τύπων δεδομένων δεν μπορεί να τροποποιηθούν (immutable)

Ιδιότητες και μέθοδοι αντικειμένων

- Ένα αντικείμενο είναι μια συλλογή ιδιοτήτων (key-value pairs). Μια ιδιότητα είναι ένα ζεύγος που αποτελείται από το όνομα της ιδιότητας (ως μεταβλητή) και την τιμή της.
- Η τιμή μιας ιδιότητας μπορεί να είναι ένα βασικός τύπος δεδομένων ή κάποιο αντικείμενο (π.χ. πίνακας, συνάρτηση, κ.λπ.)
- Όταν η τιμή μιας ιδιότητας είναι μία συνάρτηση (αντικείμενο Function), τότε η ιδιότητα αυτή καλείται μέθοδος του αντικειμένου.
- Σε ένα αντικείμενο μπορούν δυναμικά να προστεθούν, διαγραφούν και τροποποιηθούν οι ιδιότητες και οι μέθοδοι του.
- Μπορεί επίσης δυναμικά να ορισθούν ή να τροποποιηθεί ο πηγαίος κώδικας των συναρτήσεων και των μεθόδων.

Δημιουργία αντικειμένων

Τα αντικείμενα μπορούν δημιουργηθούν:

- Με την χρήση τιμών (object literals).

```
const person = {firstName:"John", lastName:"Doe", age:50};
```

- Με την χρήση συναρτήσεων (factory function)
- Με συνάρτηση δημιουργίας (function constructor)
 - Η Javascript διαθέτει έτοιμες συναρτήσεις δημιουργίας για αντικείμενα String , Number, Boolean, Object, RegExp Function και Date:

πχ.

```
let a = new Object(); // Νέο αντικείμενο τύπου Object
```

```
let b = new String(); // Νέο αντικείμενο τύπου String
```

```
let c = new Number(); // Νέο αντικείμενο τύπου Number
```

```
let d = new Boolean(); // Νέο αντικείμενο τύπου Boolean
```

Δημιουργία αντικειμένων

Με object literal

```
//creation with object literal
const nick = {
  name: "nick",
  age: 23,
  //define method
  talk: function () {
    console.log("I am " + this.name +
      " and my age is " + this.age)
  },
  //shorter definition for another name
  say() { console.log(this.name); }
};
nick.talk();
```

Με συνάρτηση δημιουργίας

```
//Function constructor
function Person(name, age) {
  this.name = name;
  this.age = age;
  this.talk = function () {
    console.log("I am " + this.name +
      " and my age is " + this.age);
  }
}

//Creation of objects with new keyword
const anna = new Person("Anna", 24);
const petros = new Person("Petros", 28);
anna.talk();
petros.talk();
```

Με συνάρτηση

```
//Factory Function
function createPerson(name, age) {
  return {
    name: name,
    age, // shortcut for: age: age,
    talk() {
      console.log("I am " + this.name +
        " and my age is " + this.age)
    }
  }
}

//creation of objects with factory function
const marios= createPerson("Marios", 27);
const eleni= createPerson("Eleni", 31);
marios.talk();
eleni.talk();
```

Η συνάρτηση δημιουργίας

Η μέθοδος new προσθέτει αυτόματα:

- μια εντολή στην αρχή της συνάρτησης για την αρχικοποίηση της μεταβλητή this (this = {})
- Μια δήλωση στο τέλος για να την επιστρέψει (return this).

Η συνάρτηση δημιουργίας

```
//Function constructor
function Person(name, age) {
  //this = {};
  this.name = name;
  this.age = age;
  this.talk = function () {
    console.log("I am " + this.name +
      " and my age is " + this.age);
  }
  //return this;
}
//Creation of objects with new keyword
const anna = new Person("Anna", 24);
const petros = new Person("Petros", 28);
```

Κώδικας που προστίθεται όταν καλείται η συνάρτηση με το new

Αντίστοιχος κώδικας με απλή συνάρτηση (factory function)

```
//Factory Function
function newPerson(name, age) {
  this.name = name;
  this.age = age;
  this.talk = function () {
    console.log("I am " + this.name +
      " and my age is " + this.age);
  }
  return this;
}

/* Creation of objects with factory function.
The method call() passes "this" as a
the first argumnet in the function */
const marios= newPerson.call({}, "Marios", 27);
const eleni= newPerson.call({}, "Eleni", 31);
```

Η ιδιότητα constructor

Όταν ένα αντικείμενο δημιουργείται με συνάρτηση δημιουργίας, η ιδιότητα constructor αναφέρει την συνάρτηση που το δημιούργησε.


```
▼ Person {name: 'Petros', age: 28, talk: f} ⓘ  
  age: 28  
  name: "Petros"  
  ► talk: f ()  
  ▼ [[Prototype]]: Object  
    ► constructor: f Person(name, age)  
    ► [[Prototype]]: Object
```

Αντικείμενο που δημιουργήθηκε με άλλο τρόπο. Η ιδιότητα constructor αναφέρει την συνάρτηση Object()

```
▼ {name: 'Marios', age: 27, talk: f} ⓘ  
  age: 27  
  name: "Marios"  
  ► talk: f ()  
  ▼ [[Prototype]]: Object  
    ► constructor: f Object()
```

Εάν η συνάρτηση δημιουργίας κληθεί κατά λάθος χωρίς το πρόθεμα new, τότε καλείται ως απλή συνάρτηση και το this αναφέρεται συνήθως στο window το οποίο και τροποποιεί!!!

```
This =  
  ► Window {window: Window, self: Window, document: document, name: 'Petros', location:  
    Location, ...}
```



objectNew.html:30

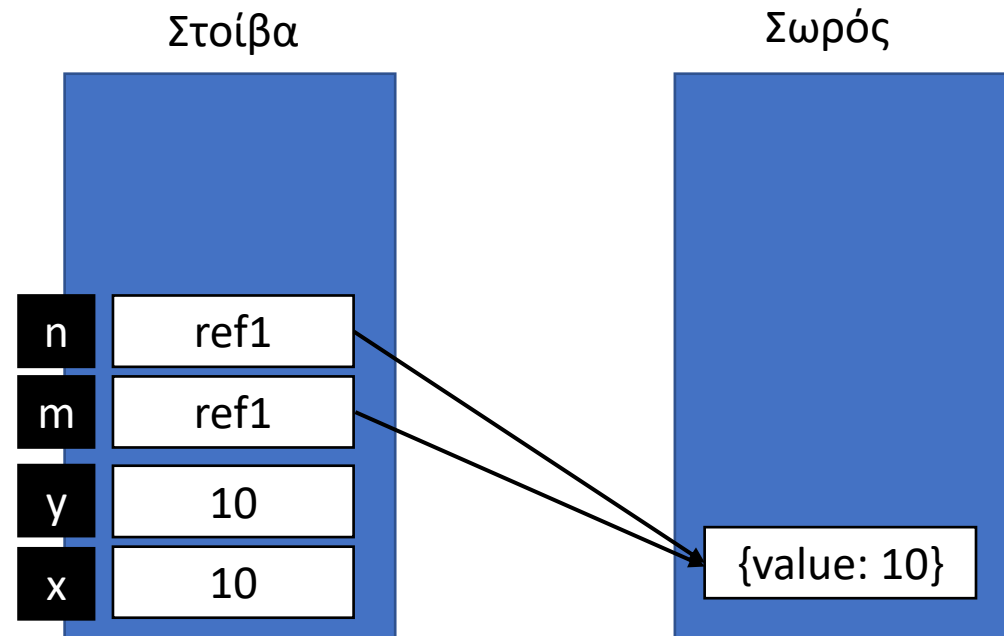
Αποθήκευση αντικειμένων - Στοίβα και σωρός

- Η μεταβλητή ενός αντικειμένου είναι ένας δείκτης στο αντικείμενο.
- Οι βασικοί τύποι δεδομένων (value type), αποθηκεύονται στην στοίβα και η κάθε μεταβλητή συσχετίζεται με διαφορετική θέση της στοίβας
- Τα αντικείμενα αποθηκεύονται στο σωρό και οι μεταβλητές που αναφέρονται σε αυτά στην στοίβα. Η μεταβλητή ενός αντικειμένου αποθηκεύει στην στοίβα την θέση (δείκτη ή αναφορά) του αντικειμένου στο σωρό.

```
let x = 10;  
let y = x;  
console.log(`x = ${x}, y = ${y}`);  
let m={value: 10};  
let n = m;  
console.log("m = ",m, " n = ", n);
```

```
x = 10, y=10
```

```
m = ▶ {value: 10} n = ▶ {value: 10}
```



Αντιγραφή αντικειμένων - Στοίβα και σωρός

- Όταν αντιγράφεται μια μεταβλητή βασικού τύπου δεδομένων (value type), γίνεται αντιγραφή της τιμής. Οι τιμές των δύο μεταβλητών είναι ανεξάρτητες μεταξύ τους και βρίσκονται αποθηκευμένες σε διαφορετικές θέσεις στην στοίβα.
- Όταν αντιγράφεται μια μεταβλητή αντικειμένου (reference type), αντιγράφεται ο δείκτης της και όχι το αντικείμενο που είναι αποθηκευμένο στο σωρό. Οι δύο μεταβλητές συσχετίζονται με το ίδιο αντικείμενο του σωρού. Οποιαδήποτε αλλαγή στον αντικείμενο επηρεάζει και τις δύο μεταβλητές.

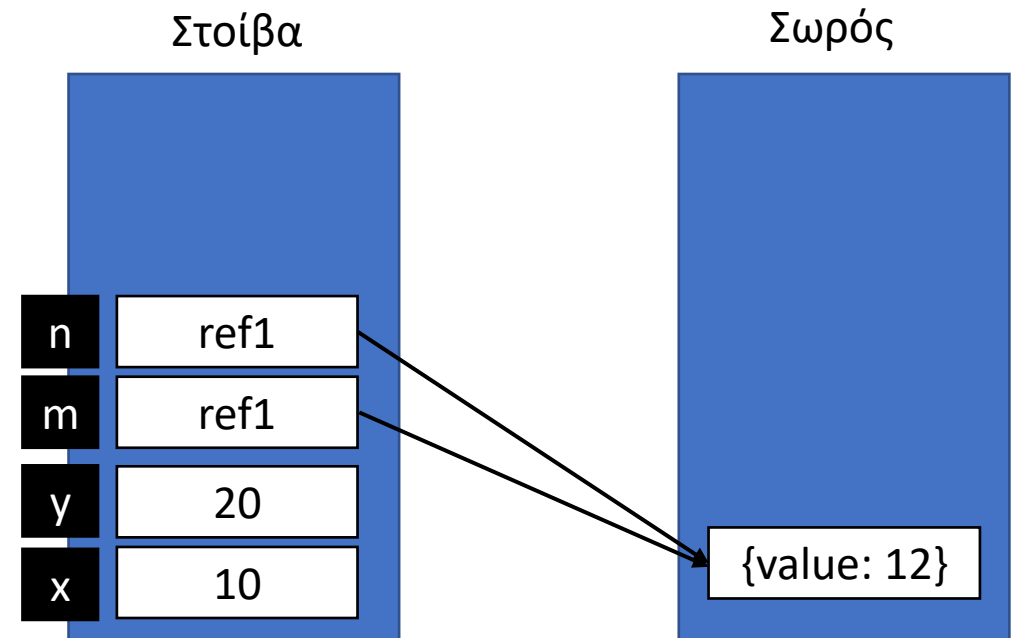
```
let x = 10;
let y = x;
console.log(`x = ${x}, y=${y}`);
y=20;
console.log(`Τιμές μετά τη αντιγραφή:
             x = ${x}, y=${y}`);
let m={value: 10};
let n = m;
console.log("m = ",m, " n = ", n);
n.value = 12;
console.log("Τιμές μετά τη αντιγραφή: m = ",
            m , " n = ", n);
```

x = 10, y=10

Τιμές μετά τη αντιγραφή: x = 10, y=20

m = ▶ {value: 10} n = ▶ {value: 10}

Τιμές μετά τη αντιγραφή: m = ▶ {value: 12} n = ▶ {value: 12}



Μέθοδοι για την αντιγραφή αντικειμένων

- Η αντιγραφή (κλωνοποίηση) των αντικειμένων μπορεί να γίνει με τις παρακάτω μεθόδους:
 - Object.assign

```
const aNewObject = Object.assign({}, anObject);
```
 - Object spread

```
const aNewObject = { ... anObject}
```
 - JSON

```
const aNewObject = JSON.parse(JSON.stringify(anObject));
```
 - structureCopy

```
const aNewObject = structureCopy(anObject);
```
- Οι δύο πρώτες μέθοδοι δημιουργούν ένα shallow copy (μόνο ενός επιπέδου) και δεν αντιγράφουν τυχόν εμφωλευμένα αντικείμενα

Παράδειγμα αντιγραφής αντικειμένων

```
let m = { value: 10 };

//copy all properties of an object with assign method
let p = Object.assign({}, m);

//spread operator to return the properties of an object
let q = { ...m };

//convert an object to json string and parse the json
//string to create a new object
let r = JSON.parse(JSON.stringify(m));

//Deep copy with structuredClone
let s = structuredClone(m);

console.log(m, p, q, r, s);
p.value = 20;
q.value = 30;
r.value = 40;
s.value = 50;
console.log(m, p, q, r, s);
```

```
▶ {value: 12} ▶ {value: 12} ▶ {value: 12} ▶ {value: 12} ▶ {value: 12}
▶ {value: 12} ▶ {value: 20} ▶ {value: 30} ▶ {value: 40} ▶ {value: 50}
```

Προσπέλαση των ιδιοτήτων αντικειμένων

Με την εντολή `for .. in`

Με την χρήση των μεθόδων `keys`
και `values` του `Object`

Με την μετατροπή τους σε JSON

```
let aUser = {
  name: "John Papas",
  userName: "john",
  email: "johnP@uth.gr",
}
console.log("display using for .. in loop");
for (let aProperty in aUser){
  console.log(`Property: ${aProperty}
    value: ${aUser[aProperty]}`);
}
console.log("display using keys / values");
console.log("Keys = ", Object.keys(aUser));
console.log("Values = ", Object.values(aUser));

console.log("display as JSON");
console.log(JSON.stringify(aUser));
```

```
display using for .. in loop
Property: name value: John Papas
Property: userName value: john
Property: email value: johnP@uth.gr
display using keys / values
Keys = ▶ (3) ['name', 'userName', 'email']
Values = ▶ (3) ['John Papas', 'john', 'johnP@uth.gr']
display as JSON
{"name":"John Papas","userName":"john","email":"johnP@uth.gr"}
```

Παράδειγμα εμφάνισης ιδιοτήτων αντικειμένου

```
function displayObject(anObject, anElementId) {
  let anElement = document.getElementById(anElementId);
  anElement.innerHTML = "";
  aContainer = document.createElement("div");
  aContainer.classList.add("objectContainer");
  for (let aProperty in anObject){
    let aLabel=document.createElement("label");
    let anInput=document.createElement("input");
    aLabel.textContent=aProperty;
    aLabel.for=aProperty;
    anInput.fname=aProperty;
    anInput.value=anObject[aProperty];
    aContainer.appendChild(aLabel);
    aContainer.appendChild(anInput);
  }
  anElement.appendChild(aContainer);
}

function testDisplay(){
  let aUser ={
    name: "John Papas",
    userName: "john",
    email: "johnP@uth.gr",
    password: "r6sLNAMd(mK3@mWk"
  }
  displayObject(aUser, "output");
}
```

```
.objectContainer {
  display: grid;
  grid-template-columns: 1fr 2fr;
  margin: 4px;
  background-color: lightgray;
}

.objectContainer *{
  margin: 4px;
  padding: 2px;
}
```

```
<body>
  <h1>Εμφάνιση Αντικειμένου</h1>
  <button onclick="testDisplay();">
    Εμφάνιση Αντικειμένου</button>
  <div id="output"></div>
</body>
```

Εμφάνιση Αντικειμένου

Εμφάνιση Αντικειμένου

name	John Papas
userName	john
email	johnP@uth.gr
password	r6sLNAMd(mK3@mWk

Πρόσθεση μεθόδων σε αντικείμενα

Πρόσθεση μεθόδου σε αντικείμενο που ήδη υπάρχει

```
//add method after the creation of object  
nick.sayName = function () {  
    console.log(`I am ${this.name}`);  
}
```

Με την αντιγραφή του από άλλο αντικείμενο.

```
//copy method from another object  
marios.sayName = nick.sayName;
```

Με τον δανεισμό της από άλλο αντικείμενο με την μέθοδο bind

```
//bind a method to another object method  
//eleni borrows nick function sayName  
let eleniSayName = nick.sayName.bind(eleni);  
eleniSayName();
```

```
//or with IIFE  
//Immediate Invocation Function Execution  
nick.sayName.bind(eleni)();
```

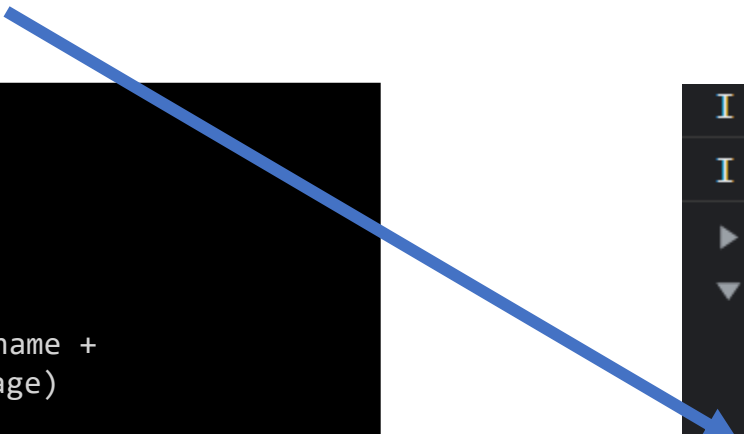
Prototypes: Η μέθοδος Object.create

- Η μέθοδος create του αντικειμένου Object δημιουργεί ένα «παρόμοιο» αντικείμενο με ίδιες ιδιότητες και μεθόδους. Το νέο αντικείμενο χρησιμοποιεί ως πρότυπο (prototype) το αντικείμενο από το οποίο προήλθε.

```
//creation with object literal
const nick = {
  name: "nick",
  age: 23,
  //define method
  talk() {
    console.log("I am " + this.name +
      " and my age is " + this.age)
  },
};
//Creation of a similar object
const mary = Object.create(nick);
mary.age=45;
mary.name="mary";

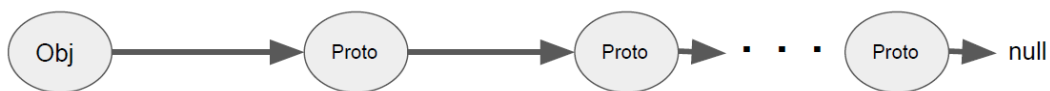
nick.talk();
mary.talk();
console.log(nick,mary);
```

```
I am nick and my age is 23
I am mary and my age is 45
▶ {name: 'nick', age: 23, talk: f}
▼ {age: 45, name: 'mary'} ⓘ
  age: 45
  name: "mary"
  ▼ [[Prototype]]: Object
    age: 23
    name: "nick"
    ▶ talk: f talk()
    ▶ [[Prototype]]: Object
```



Πρότυπα (prototypes) αντικειμένων

- Όλα τα αντικείμενα της Javascript έχουν ένα πρότυπο (prototype) από το οποίο κληρονομούν τις ιδιότητες και μεθόδους του.
- Στην κορυφή της ιεραρχίας βρίσκεται το αντικείμενο Object του οποίου το prototype είναι null. Όλα τα υπόλοιπα αντικείμενα, άμεσα ή έμμεσα, κληρονομούν ιδιότητες και μεθόδους από αυτό.



```
▼ {} ⓘ  
  ▼ [[Prototype]]: Object  
    ▶ constructor: f Object()  
    ▶ hasOwnProperty: f hasOwnProperty()  
    ▶ isPrototypeOf: f isPrototypeOf()  
    ▶ propertyIsEnumerable: f propertyIsEnumerable()  
    ▶ toLocaleString: f toLocaleString()  
    ▶ toString: f toString()  
    ▶ valueOf: f valueOf()  
    ▶ __defineGetter__: f __defineGetter__()  
    ▶ __defineSetter__: f __defineSetter__()  
    ▶ __lookupGetter__: f __lookupGetter__()  
    ▶ __lookupSetter__: f __lookupSetter__()  
    ▶ __proto__: (...)  
    ▶ get __proto__: f __proto__()  
    ▶ set __proto__: f __proto__()
```

Ένα νέο αντικείμενο {} έχει ως prototype το Object και κληρονομεί τις ιδιότητες του και τις μεθόδους του, όπως toString και constructor

Κληρονομικότητα (prototypical inheritance)

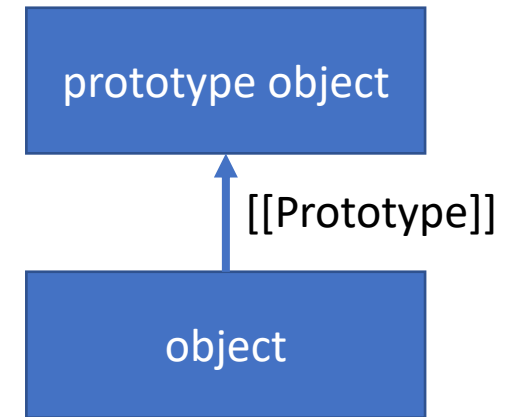
- Ο μηχανισμός κληρονομικότητας της Javascript διαφέρει από τις άλλες γλώσσες προγραμματισμού.
- Η κληρονομικότητα της Javascript στηρίζεται στην ιδιότητα prototype των αντικειμένων.
- Αν και πολλοί προγραμματιστές θεωρούν ότι αυτός ο τρόπος κληρονομικότητας είναι μια αδυναμία της Javascript, στην πράξη προσφέρει μεγαλύτερη ευελιξία και περισσότερες δυνατότητες από την κλασική κληρονομικότητα η οποία στηρίζεται σε κλάσεις αντικειμένων.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain

- Η κληρονομικότητα της Javascript είναι δυναμική. Μπορεί να αλλάξει δυναμικά κατά την εκτέλεση και να τροποποιηθούν τα πρότυπα των αντικειμένων.
- Δεν υποστηρίζεται πολλαπλή κληρονομικότητα
- Αν και στις τελευταίες εκδόσεις έχει γίνει η εισαγωγή κλάσεων αντικειμένων στην Javascript, οι κλάσεις δεν χρησιμοποιούνται για την κληρονομικότητα

Ο μηχανισμός της κληρονομικότητας

- Ένα αντικείμενο διαθέτει, εκτός από τις δικές του ιδιότητες/μεθόδους, και τις ιδιότητες/μεθόδους του ορίζει το πρότυπο του. Το πρότυπο με την σειρά του διαθέτει και τις ιδιότητες/μεθόδους του δικού του πρότυπου, κ.ο.κ. μέχρι να εξαντληθεί η ιεραρχία στο αντικείμενο Object του οποίου το πρότυπο είναι null.
- Όταν η Javascript αναζητά μια ιδιότητα, αν δεν την βρει στις ιδιότητες του αντικειμένου, την αναζητά στην ιεραρχία των προτύπων του.



Προσθήκη μεθόδων και ιδιοτήτων στο prototype

Πρόσθεση νέας μεθόδου στο prototype. Η μέθοδος προστίθεται σε όλα τα αντικείμενα Person

Πρόσθεση νέας ιδιότητας στο prototype κοινή μόνο για ανάγνωση σε όλα τα αντικείμενα.

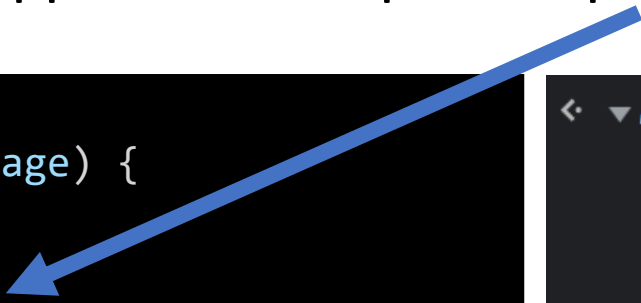
```
//Function constructor
function Person(name, age) {
  this.name = name;
  this.age = age;
  this.talk = function () {
    console.log("I am " + this.name +
      " and my age is " + this.age);
  }
}

//Creation of objects with new keyword
const anna = new Person("Anna", 24);
const petros = new Person("Petros", 28);
//Add a new method for all Persons
Person.prototype.log = function () {
  console.log(this);
}
//Add a new property to all Persons
Person.prototype.id = 0;
console.log(anna.id);
anna.log();
petros.log();
```

Ιδιωτικές ιδιότητες αντικειμένων

Μία μεταβλητή της συνάρτησης δημιουργίας, αν δεν ορισθεί ως ιδιότητα του αντικειμένου (πχ. `this.age = age`) αλλά χρησιμοποιείται από κάποια από τις μεθόδους του, τότε γίνεται ιδιωτική μεταβλητή στην οποία υπάρχει πρόσβαση μόνο από την συνάρτηση δημιουργίας

```
//Function constructor
function Person(name, age) {
  this.name = name;
  //Private property
  let _age = age;
  this.talk = function () {
    console.log("I am " + this.name +
      " and my age is " + _age);
  }
}
const anna = new Person("Anna", 24);
const petros = new Person("Petros", 28);
anna.talk();
petros.talk();
```



```
Person {name: 'Anna', talk: f} ⓘ
  name: "Anna"
  talk: f ()
    arguments: null
    caller: null
    length: 0
    name: ""
  ▶ prototype: {constructor: f}
  [[FunctionLocation]]: objectGettersSetters.
  ▶ [[Prototype]]: f ()
  ▼ [[Scopes]]: Scopes[3]
    ▶ 0: Closure (Person) {_age: 24}
    ▶ 1: Script {anna: Person, petros: Person}
    ▶ 2: Global {0: Window, window: Window, sel
  ▶ [[Prototype]]: Object
```

Getter / Setters ιδιοτήτων αντικειμένων

Μία μέθοδος του αντικειμένου μπορεί να έχει πρόσβαση σε μία ιδιωτική μεταβλητή. Μπορούμε επίσης με τον ορισμό getters ή και setters να ορίσουμε ένα εικονικό όνομα ιδιότητας για την πρόσβαση στην μεταβλητή


```
//Function constructor
function Person(name, age) {
  this.name = name;
  //Private property
  let _age = age;
  //Define getter /setter for object
  Object.defineProperty(this, "age", {
    get: function(){
      return _age;
    },
    set: function(value){
      _age = value;
    }
  });
  //method accessing private var
  this.getAge = () => age;
  this.talk = function () {
    console.log("I am " + this.name +
      " and my age is " + _age);
  }
}
```

```
> anna.getAge()
< 24
> anna.age=23
< 23
> anna.age
< 23
```

Παράδειγμα κληρονομικότητας αντικειμένων

Τα αντικείμενα που δημιουργούνται με `new Student` κληρονομούν τις ιδιότητες και μεθόδους των αντικειμένων `Person`

```
function Student(name, age, school) {  
    Person.call(this, name, age);  
    this.school = school;  
}  
  
Student.prototype = Person.prototype;  
Student.prototype = new Person();  
Student.prototype.constructor = Student;  
  
let jim = new Student("Jim", 19,  
    "Τμήμα Πληροφορικής");
```



```
//Function constructor  
function Person(name, age) {  
    this.name = name;  
    this.age = age;  
    this.talk = function () {  
        console.log("I am " + this.name +  
            " and my age is " + this.age);  
    }  
}
```

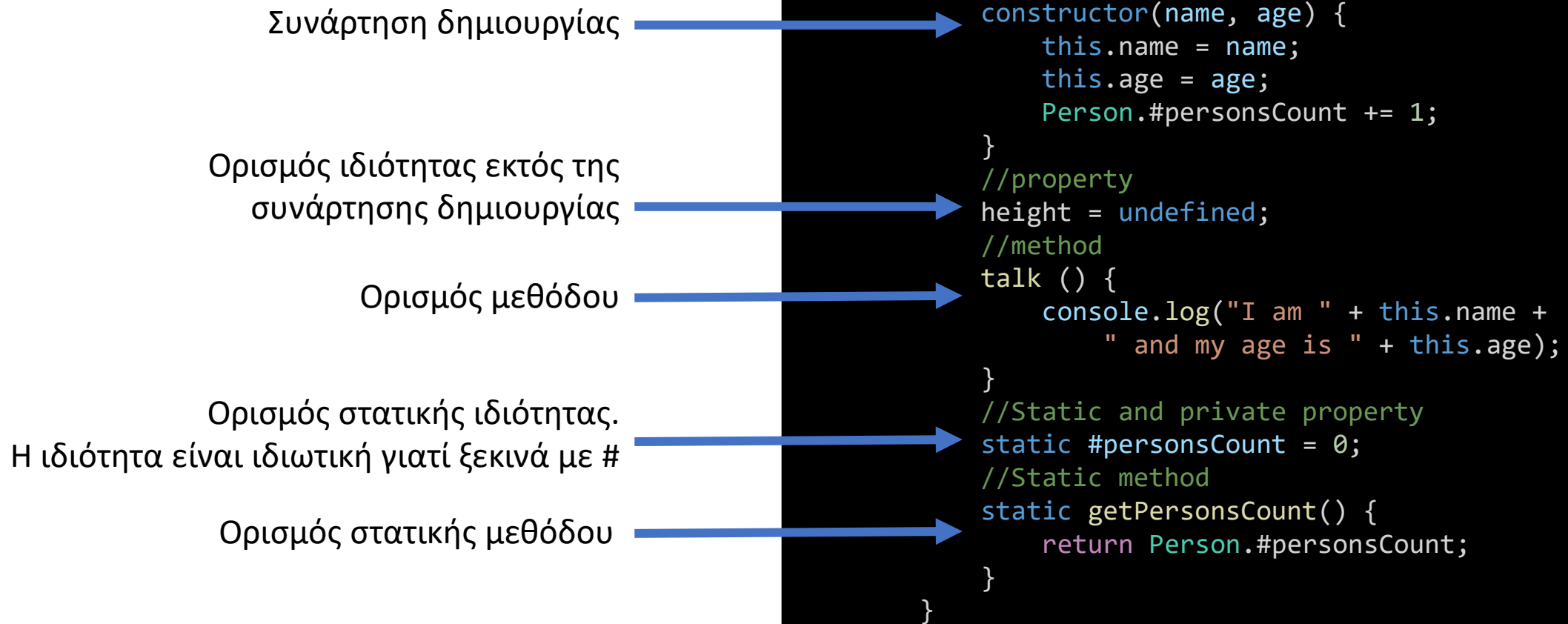
Κλάσεις αντικειμένων

Η νέα έκδοση ES6 το 2015, εισήγαγε τις κλάσεις στην Javascript

Με τις κλάσεις μπορούμε να ορίσουμε:

- Κληρονομικότητα (extends)
- Μέθοδο δημιουργίας (constructor)
- Μεθόδους και ιδιότητες των αντικειμένων
- Ιδιωτικές (private) μεθόδους και ιδιότητες με την χρήση στην αρχή της ονομασίας τους τον χαρακτήρα #
- Getter και Setters
- Στατικές μεθόδους και ιδιότητες οι οποίες είναι προσβάσιμες με την χρήση του ονόματος της κλάσης.

Παράδειγμα κλάσης Person



Παράδειγμα κληρονομικότητας κλάσης Student

Ορισμός κλάσης Student η οποία κληρονομεί από την κλάση Person με την δήλωση extends Person

Ορισμός ιδιωτικής ιδιότητας (ξεκινά με #)

Ορισμός getter

Ορισμός setter ο οποίος ελέγχει την τιμή του βαθμού πριν την καταχωρήσει

Δημιουργία σφάλματος σε περίπτωση λάθους βαθμού

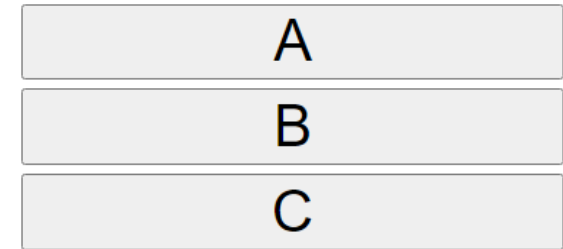
```
//Definition of a subclass of Student
class Student extends Person {
  constructor(name, age, school) {
    //initialize superclass
    super(name, age);
    this.school = school;
  }
  //private property
  #grade = undefined;
  //Getter settter for grade
  get grade() {
    //report that someone accessed the grade
    console.log("returning grade " + this.#grade);
    return this.#grade;
  }
  set grade(x) {
    //check grade
    if (x >= 0 && (x <= 10)) {
      console.log("changing grade to " + x);
      this.#grade = x;
    }
    else {
      //throw error if invalid grade
      throw ("Invalid grade " + x);
    }
  }
}
```

Παράδειγμα κλάσης πλήκτρου

Παράδειγμα κλάσης πλήκτρου με μέθοδο δημιουργίας η οποία δημιουργεί ένα πλήκτρο και το τοποθετεί σε ένα html στοιχείο

```
class Button {
  constructor(containerId, textContent) {
    const buttonContainer =
      document.querySelector('#'+containerId);
    this.parent = buttonContainer;
    this.textContent = textContent;
    const button = document.createElement('button');
    button.textContent = textContent;
    //binds this to the method on click
    this.onClick = this.onClick.bind(this);
    button.addEventListener('click',
      this.onClick);
    this.parent.append(button);
  }
  onClick() {
    console.log('clicked: ' + this.textContent);
  }
}
const button1 = new Button('toolbar', 'A');
const button2 = new Button('toolbar', 'B');
const button3 = new Button('toolbar', 'C');
```

Παράδειγμα κλάσης πλήκτρου



```
<body>
  <h1>Παράδειγμα κλάσης πλήκτρου</h1>
  <div id="toolbar"></div>
</body>
```


Αναφορές

- <https://developer.mozilla.org/en-US/docs/Learn/JavaScript>
- <https://www.w3schools.com/js/default.asp>
- https://www.w3schools.com/js/js_examples.asp
- <https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/#basic-javascript>
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance and the prototype chain](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain)
- <https://www.tutorialsteacher.com/javascript/prototype-in-javascript>
- [Object-oriented Programming in JavaScript: Made Super Simple | Mosh](#)