ΕΙΣΑΓΩΓΗ ΣΤΗΝ JAVASCRIPT

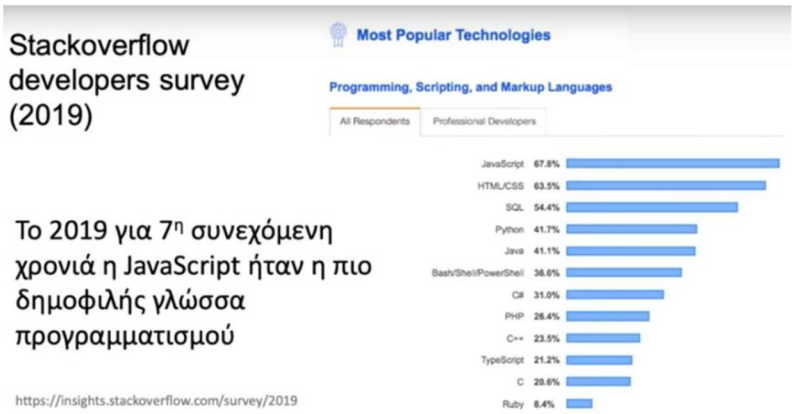
Εισαγωγή, ιστορικά στοιχεία

Εισαγωγή

• Η JavaScript είναι μια διερμηνευόμενη γλώσσα υψηλού επιπέδου που ακολουθεί τις προδιαγραφές ECMAScript.
• Δημιουργήθηκε αρχικά το 1995 από τον μηχανικό της Netscape **Brendan Eich.**
• Η JavaScript μαζί με την HTML και CSS, είναι οι βασικές τεχνολογίες για ανάπτυξη εφαρμογών του παγκόσμιου ιστού.
• Όλοι οι φυλλομετρητές έχουν μια μηχανή JavaScript που επιτρέπει εκτέλεση κώδικα JS.
• Η JS χρησιμοποιείται ακόμη για εφαρμογές στον εξυπηρετητή με το node.js
https://www.ecma-international.org/ecma-262/5.1

Κύρια χαρακτηριστικά

• Η JavaScript χαρακτηρίζεται ως δυναμική γλώσσα με ασθενή διαχείριση τύπων δεδομένων στηρίζεται σε πρότυπα (prototypes) και ακολουθεί πολλαπλά
παραδείγματα (event driven, functional, object oriented)
• Διαθέτει APIs για διαχείριση κειμένου, πινάκων, ημερομηνιών τυπικές εκφράσεις, και το DOM της HTML.
• Δεν περιλαμβάνει I/O, για δικτύωση, μόνιμη αποθήκευση, ή γραφική διεπαφή.



Η εξέλιξη στη χρήση της JavaScript

Μέχρι το 1999 η JS είχε κύρια χρήση να παρέχει διαδραστικότητα σε έγγραφα HTML/XHTML:
-Να ελέγχεται η εγκυρότητα δεδομένων
-Να επιτρέπει αντίδραση σε γεγονότα που προκαλεί ο χρήστης (με δείξη ή πληκτρολόγιο)
-Να διαβάζει και να τροποποιεί HTML elements
-Να αναγνωρίζει το περιβάλλον του χρήστη
-Να δημιουργεί cookies
Με την έλευση της τεχνολογίας AJAX το 1999, (ασύγχρονης κλήσης στον εξυπηρετητή από μια ιστοσελίδα), η γλώσσα άλλαξε χρήση και άρχισε να αναπτύσσεται ραγδαία

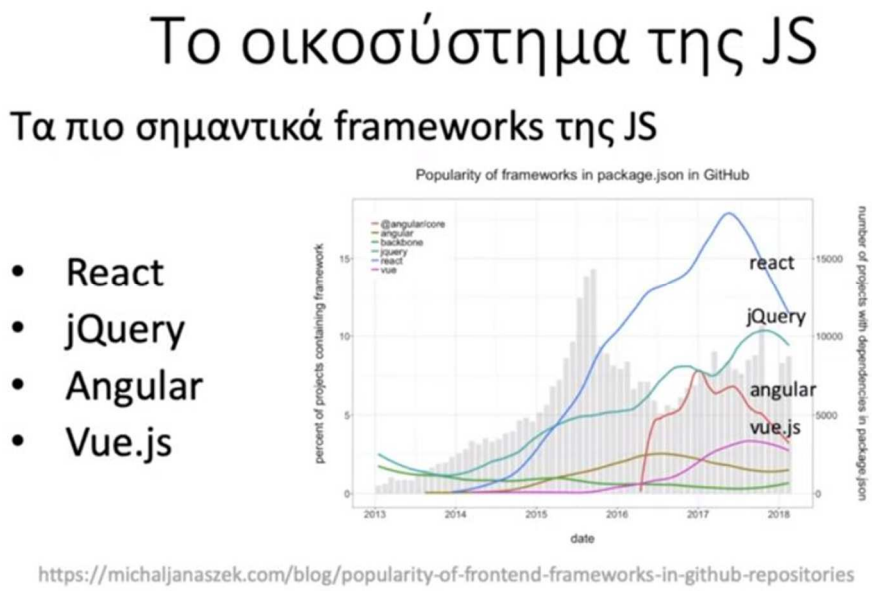
Σήμερα η κύρια χρήση της είναι να μεταφέρει τμήμα της λογικής της εφαρμογής στο φυλλομετρητή, και έχει τα χαρακτηριστικά μιας πλήρους γλώσσας προγραμματισμού. Αυτό οδήγησε στην ανάπτυξη βιβλιοθηκών JavaScript (τοοικοσύστημα της JS)

Εκδόσεις JavaScript/ECMAScript (ES)

ES3 (1999): regular expression literals, χειριστής εξαιρέσεων try/catch
ES5 (2009): map/reduce/filter/forEach, υποστήριξη JSON,getters/setters reflection
ES6 (2015): promises (ασύγχρονη εκτέλεση), modules, classes,block scope variables (let), arrow functions => , template literal,spread, parameter default values, symbols, generator functions
Νεότερες εκδόσεις **ES7, ES8** χωρίς σημαντικές αλλαγές από την ES6, **ετήσιες αναθεωρήσεις**

Το οικοσύστημα της JavaScript

Η κύρια χρήση της JS είναι σήμερα: να πραγματοποιεί ασύγχρονες κλήσεις στον server, να επεξεργάζεται το Μοντέλο αντικειμένων του εγγράφου (DOM) και να ανταποκρίνεται σε γεγονότα που προκαλεί ο χρήστης. Επειδή η JS δεν παρείχε εύκολες δομές για τα παραπάνω, άρχισε η δημιουργία βιβλιοθηκών που επεκτείνουν τη λειτουργία της. Η πιο σημαντική τέτοια βιβλιοθήκη είναι η jQuery (2006), αν και σήμερα τα νεότερα πρότυπα της JS έχουν υλοποιήσει τις περισσότερες λειτουργίες της jQuery.
https://www.imaginawcloud.com/blog/a+javascript-ecosystem-overview/

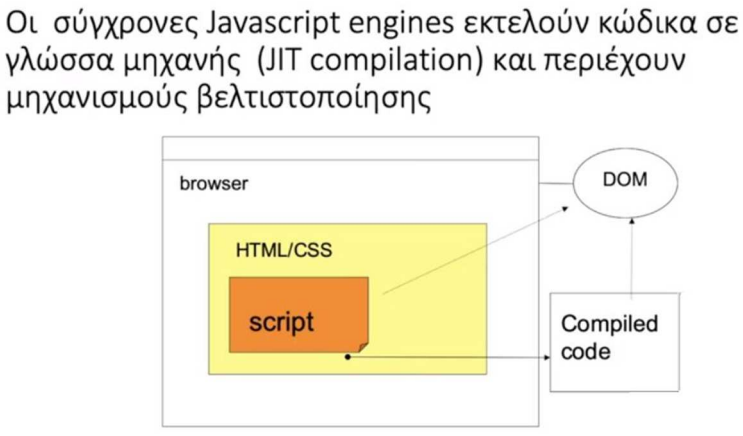


Javascript engines

Ο φυλλομετρητής για κάθε ιστοσελίδα ξεκινάει δύο διεργασιες η μία είναι η **rendering engine** που σχεδιάζει την ιστοσελίδα και η άλλη είναι η **javascript engine** που εκτελεί τον κώδικα javascript
Οι σύγχρονοι φυλλομετρητές δεν διερμηνεύουν την JavaScript αλλά την μεταγλωττίζουν Just In Time (JIT compilation)

Οι κύριες JS engines

- V8: στον Chrome, Node.js, και V8.NET.
- SpiderMonkey: στον Firefox.
- JavaScriptCore:στον Safari. (γνωστός ως Nitro)



Ενσωμάτωση JS σε κώδικα HTML

internal JavaScript

Ως κώδικας τυπικά τοποθετείται εντός της περιοχής **<head>...</head>** ενός εγγράφου, προσβάσιμος μόνο από αυτό με την χρήση του **<script>**

```
<script>
//κώδικας Javascript
</script>
```

external JavaScript(προτεινόμενος τρόπος**)**
Το έγγραφο μας συνδέεται με εξωτερικό αρχείο

```
<script src="scriptfile.js" async></script>
```

async γνώρισμα για ασύγχρονη εκτέλεση **πριν** την ολοκλήρωση ανάλυσης HTML

defer **μετά** τη φόρτωση HTML ή άλλου κώδικα JS

inline JavaScript(λιγότερο χρησιμοποιούμενος και όχι προτεινόμενος)

```
<button onclick="createParagraph()">Click me!</button>
```

Γενικά η χρήση JS σε γραμμές κώδικα HTML **δεν είναι καλή πρακτική**
Θα μπορούσε να προστεθεί ένας **event listener** στα πλήκτρα ενός κώδικα HTML ως εξής:

```
const buttons = document.querySelectorAll('button');
for(let i= 0; i < buttons.length ; i++) {
        buttons[i].addEventListener('click',createParagraph);
```

JAVASCRIPT που εκτελείται μετά το φόρτωμα της σελίδας

```
<script>
function func1(){
....
}
</script>
```

<body onload="func1()"> **←μετά το φόρτωμα της σελίδας**

ΑΝΤΙΘΕΤΑ:η εντολή :window.onload αφορά όχι μόνο το φόρτωμα της σελίδας αλλά και επίσης φόρτωμα των εικόνων και άλλων βοηθητικών αρχείων

-Αν χρησιμοποιούμε τη JavaScript **για χειρισμό στοιχείων ενός DOM** Document Obiect Model, ο κώδικας JavaScript θα πρέπει **να φορτωθεί μετά** την ολοκλήρωση ανάλυσης της HTML.

• Λύση για internal JavaScript:

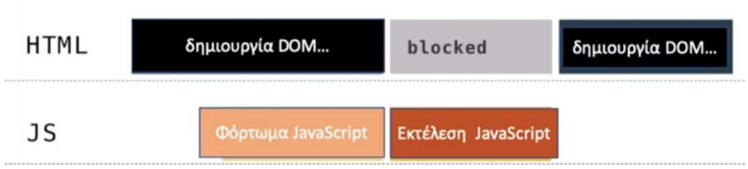
```
document.addEventListener("DOMContentLoaded", function() {..});
```

• Για external JavaScript μπορούμε να χρησιμοποιήσουμε τις λέξεις κλειδιά **defer ή async**

async

<script src="scriptfile.js" async></script>

Στην περίπτωση αυτή ο κώδικας javascript εκτελείται στην πρώτη ευκαιρία **μετά την ολοκλήρωση φόρτωσης του** και πριν το γεγονός **load του window**.Δεν τηρείται σειρά και είναι πιθανόν να διακοπεί η δημιουργία DOM.



defer

<script src="scriptfile.js" defer></script>

Στην περίπτωση αυτή ο κώδικας javascript εκτελείται μετά την ολοκλήρωση φορτώματος της HTML ή άλλου script αλλά πριν το γεγονός **DOMContentLoaded.**

ΔΙΑΔΟΧΙΚΟ ΦΟΡΤΩΜΑ ΚΩΔΙΚΑ JS

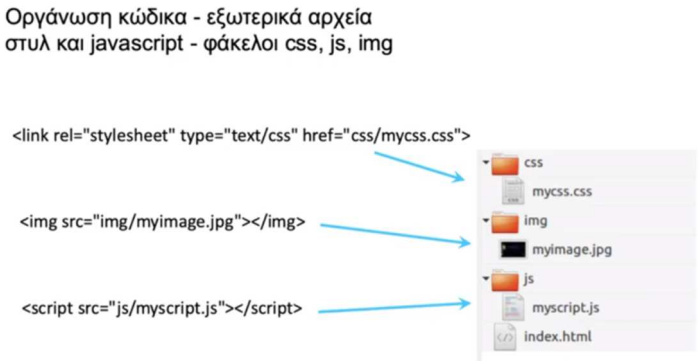
Αν κάποιος κώδικας JS πρέπει να περιμένει ή για άλλο τμήμα κώδικα ή για το φόρτωμα της HTML τότε μπορούμε να χρησιμοποιήσουμε το γνώρισμα **defer** στην ετικέτα <script> **με τη σειρά με την οποία επιθυμούμε την εκτέλεση τους**.

<script src="js/script2.js" defer></script>

<script src="js/script3.js" defer></script>

Ο κώδικας script2.js θα εκτελεστεί **πριν** το script3.js

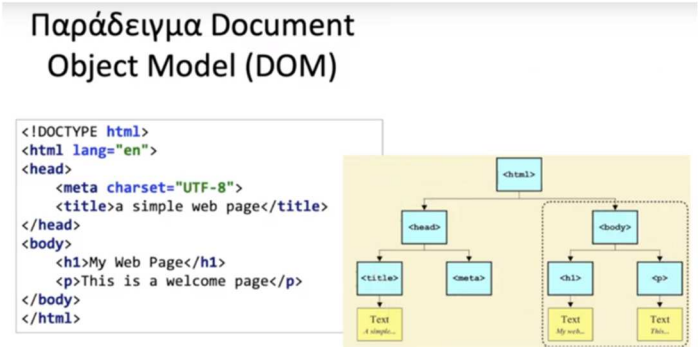
ΕΠΙΛΟΓΕΣ ΦΟΡΤΩΣΗΣ ΚΩΔΙΚΑ Javascript
-Γενικά τοποθετούμε το **<script>** μέσα στο <head> και χρησιμοποιούμε τα γνωρίσματα **async** και **defer** ώστε η JS να μην μπλοκάρει το κατέβασμα της σελίδας.
-Χρησιμοποιούμε **<script async…>** αν το περιεχόμενο της σελίδας είναι **ανεξάρτητο** από τον κώδικα Javascript
-Χρησιμοποιούμε **<script defer…>** αν επιθυμούμε ο κώδικας να τρέξει **μετά** την ολοκλήρωση φορτώματος της HTML
-Εναλλακτικά μπορούμε να χρησιμοποιήσουμε απλά το **<script>** αλλά στο τέλος του </body> ώστε **να μην μπλοκάρουμε** το κατέβασμα της σελίδας.



ΔΙΑΣΥΝΔΕΣΗ JS ΜΕ ΤΟ DOM

το web API της JS (navigator / window / document)
-ο **navigator** αντιπροσωπεύει την κατάσταση και την ταυτότητα του φυλλομετρητή (user agent).
-Το **window** είναι το παράθυρο που φορτώνεται η ιστοσελίδα.Από αυτό μπορούμε να πάρουμε το μέγεθος του παραθύρου(window.innerWidth ,window.innerHeight..κλπ.
-Το **document** (που αντιπροσωπεύεται από την DOM στον φυλλομετρητή) είναι η ιστοσελίδα που έχει φορτωθεί στο παράθυρο, περιέχει ιεραρχία στοιχείων , <html>

Το Document Object Model (DOM)
-Το DOM είναι η προγραμματιστική διεπαφή του εγγράφου HTML
-Αναπαριστά το έγγραφο ως μια ιεραρχία από κόμβους, τα οποία είναι αντικείμενα τα οποία το πρόγραμμα μπορεί να μεταβάλει (δομή , στυλ , περιεχόμενο)



ΚΟΜΒΟΙ DOM
element node: Ένα στοιχείο του DOM
root node:Ο κορυφαίος κόμβος στο δέντρο, στην περίπτωση της HTML είναι πάντα ο κόμβος <html>
child node(παιδί):Ένας κόμβος απευθείας μέσα σε άλλο κόμβο
descendant node(απόγονος):Ένας κόμβος οπουδήποτε μέσα σε άλλο κόμβο.
parent node(γονέας):Ο κόμβος μέσα στον οποίο βρίσκεται ο κόμβος.
sibling nodes(αδέρφια):Κόμβοι που είναι στο ίδιο επίπεδο του δέντρου DOM
text node:Ένας κόμβος που περιέχει κείμενο

ΕΠΙΛΟΓΗ ΣΤΟΙΧΕΙΩΝ DOM

-document.querySelector()

επιτρέπει την επιλογή στοιχείων του DOM με χρήση επιλογέων CSS

π.χ. Document.querySelector('a') θα επιλέξει **το πρώτο στοιχείο <a>** του εγγράφου

-document.querySelectorAll()

ταιριάζει σε κάθε στοιχείο που ικανοποιεί τον επιλογέα CSS και αποθηκεύει τα αποτελέσματα σε ένα αντικείμενο τύπου array που ονομάζεται **NodeList**

παραδείγματα

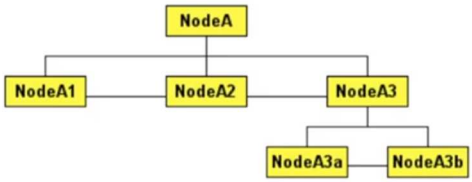
element.setAttribute(attribute,value)	δίνω τιμή σε γνώρισμα του στοιχείου
element.getAttribute(attribute)	παίρνω την τιμή ενός γνωρίσματος
element.removeAttribute(attribute)	αφαιρώ ένα γνώρισμα
element.innerHTML	το περιεχόμενο του στοιχείου
element.textContent	το κείμενο του στοιχείου
element.style	οι προδιαγραφές CSS του στοιχείου
element.style.fontSize='20px';	ορίζω μία ιδιότητα στυλ CSS
document.getElementById(id)	επιλέγω με βάση το id
document.getElementsByClassName(class)	επιλέγω με βάση την κλάση
document.getElementsByTagName(tagname)	επιλέγω με βάση την ετικέτα

(τα τελευταία 3 έχουν ενσωματωθεί στο querySelector)

ΠΡΟΣΟΧΗ: επειδή η javascript δεν υποστηρίζει την - ,το font-size της CSS π.χ. γίνεται fontSize

παραδείγματα

```
NodeA.firstChild = NodeA1
NodeA.lastChild = NodeA3
NodeA.childNodes.length = 3
NodeA.childNodes[0] = NodeA1
NodeA.childNodes[1] = NodeA2
NodeA.childNodes[2] = NodeA3
NodeA1.parentNode = NodeA
NodeA1.nextSibling = NodeA2
NodeA3.prevSibling = NodeA2
NodeA3.nextSibling = null
NodeA.lastChild.firstChild = NodeA3a
NodeA3b.parentNode.parentNode = NodeA
```




```
let link = document.querySelector('a');
link.textContent = 'University of Patras';
link.href = 'https://www.upatras.gr';
```

ΔΗΜΙΟΥΡΓΙΑ - ΔΙΑΓΡΑΦΗ ΣΤΟΙΧΕΙΩΝ createElement , appendChild, removeChild

-Έστω ότι θέλουμε **να δημιουργήσουμε** ένα κόμβο <p> κάτω από τον κόμβο <section>

```
let sect=document.querySelector('section');
let para=document.createElement('p');
para.textContent='Thank you for visiting!';
sect.appendChild(para);
```

-Έστω ότι θέλουμε **να διαγράψουμε** ένα κόμβο

```
sect.removeChild(linkPara);//to remove a child of element sect
```

ΠΩΣ ΕΠΙΚΟΙΝΩΝΕΙ Η JS ΜΕ ΤΟΝ ΧΡΗΣΤΗ

-από την **κονσόλα** του browser: console.log('hello')

-**alert box**: window.alert('hello')//μήνυμα σε αναδυόμενο παράθυρο

-**prompt**: window.prompt('name:') //εισαγωγή πληροφορίας

-Με **τροποποίηση ενός element**: innerHTML('hello')

-Ή με αλλαγή του **textContent**: element.textContent='hello'

-εισαγωγή στοιχείων **σε φόρμα**:

όνομα : <input id="in1' type=text>

```
console.log(document.querySelector('#in1').value);
```

ΣΥΝΤΑΞΗ JS - ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

<u>ονόματα μεταβλητών</u>
-οι χαρακτήρες (0..9,a..z,A..Z) και η κάτω παύλα (_)
-υπάρχει **διαφορά μεταξύ κεφαλαίων και μικρών** χαρακτήρων
-Τα ονόματα **αρχίζουν** με λατινικό χαρακτήρα
-Τα ονόματα συνήθως ακολουθούν την **σήμανση καμήλας** π.χ. camelNotation,yesOrNo
-Δεν πρέπει να χρησιμοποιούνται λέξεις κλειδιά var,let,function,false,true,const,name…
ως ονόματαμεταβλητών
-Τα ονόματα μεταβλητών θα πρέπει να φέρουν νόημα

σύμφωνα με το τελευταίο πρότυπο της Javascript τα ονόματα μεταβλητών μπορούν να περιέχουν **οποιονδήποτε Unicode χαρακτήρα**,άρα και Ελληνικούς χαρακτήρες αν και αυτό **δεν προτείνεται** σαν καλή πρακτική
-Επίσης ο **αρχικός χαρακτήρας** μπορεί να είναι οποιοσδήποτε αλφαβητικός χαρακτήρας **a..z ή A..Z ή κάτω παύλα ή ο χαρακτήρας $** άρα αποδεκτά είναι και τα:
let $1=5;
let _1=10;

ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ
-Δεν απαιτείται η δήλωση τύπου δεδομένων (π.χ. int i=5;)
-Οι κύριοι τύποι δεδομένων είναι:
numbers / booleans / strings / arrays / objects

<u>παραδείγματα</u>
```
let myAge=17;
let hollowMen='Not with a bang';
hollowMen+='but a whimper';
let iAmAlive=true;
let test=6<3;
let myNumberArray=[10,15,40];
let dog={name:'Spot', breed:'Dalmatian'};
```

Number
let width=window.innerWidth; (ακέραιος)
let pi=3.14; (πραγματικός)
String
let hello='καλημέρα';
Boolean
let check=true;//or false
Object(αντικείμενο)
let node=document.querySelector("#n1");
Array(πίνακας-λίστα)
let l=document.querySelectorAll("a");
<u>Ο τύπος δεδομένων μιας μεταβλητής μπορεί να αλλάξει</u>
π.χ. let myNumber='500';// a string
typeof myNumber;
> "string"

myNumber=500;//this is a number
typeof myNumber;
> "number"

ΕΙΔΙΚΟΙ ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

-undefined

Αποδίδεται στις μεταβλητές,όταν αυτές δηλώνονται χωρίς αρχική τιμή

-null

Είναι ένα αντικείμενο.Αποδίδεται στις μεταβλητές,που έχουν μεν δηλωθεί, αλλά έχουν κενή τιμή ή αν μια διεπαφή δεν επιστρέψει τιμή

ΤΕΛΕΣΤΗΣ ΑΝΑΘΕΣΗΣ ΤΙΜΗΣ

= π.χ. total=100;
+= π.χ. total+=shipping →total=total+shipping;
-= π.χ. total-=shipping →total=total-shipping;
= π.χ. total=shipping →total=total*shipping;
/= π.χ. total/=shipping →total=total/shipping;

ΤΕΛΕΣΤΗΣ ΣΥΝΘΕΣΗΣ ΣΥΜΒΟΛΟΣΕΙΡΩΝ

π.χ. PP="Πανεπιστήμιο"+" Πατρών"; →PP="Πανεπιστήμιο Πατρών"

ΑΡΙΘΜΗΤΙΚΟΙ ΤΕΛΕΣΤΕΣ

Παράδειγμα	Περιγραφή
a + b	Πρόσθεση
a - b	Αφαίρεση
a * b	Πολλαπλασιασμός
a / b	Διαίρεση
a % b	Υπόλοιπο διαίρεσης

ΤΕΛΕΣΤΕΣ ΑΥΞΗΣΗΣ-ΜΕΙΩΣΗΣ

Παράδειγμα	Περιγραφή
a++	Απόδοση τιμής και αύξηση κατά 1
a--	Απόδοση τιμής και μείωση κατά 1
++a	Αύξηση τιμής κατά 1 και απόδοση τιμής στη συνέχεια
--a	Μείωση τιμής κατά 1 και απόδοση τιμής στη συνέχεια
a += b	Αύξηση της τιμής του a κατά b
a -= b	Μείωση της τιμής του a κατά b

ΤΕΛΕΣΤΕΣ ΠΡΑΞΕΩΝ ΣΕ ΔΥΑΔΙΚΑ ΨΗΦΙΑ

Παράδειγμα	Περιγραφή
a & b	Bitwise AND ανάμεσα στις μεταβλητές
a | b	Bitwise OR ανάμεσα στις μεταβλητές
a ^ b	Bitwise XOR ανάμεσα στις μεταβλητές
~a	Bitwise NOT της μεταβλητής
a << b	Αριστερή ολίσθηση κατά b θέσεις των bit της μεταβλητής a
a >> b	Δεξιά ολίσθηση κατά b θέσεις των bit της μεταβλητής a

ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ

Παράδειγμα	Περιγραφή
!a	Λογική αντιστροφή της τιμής της μεταβλητής
a && b	Λογικό AND
a || b	Λογικό OR

ΤΕΛΕΣΤΕΣ ΣΥΓΚΡΙΣΗΣ

Παράδειγμα	Περιγραφή
a == b	Έλεγχος ισότητα τιμής
a === b	Έλεγχος ισότητα τιμής και τύπου δεδομένων (strict version)
a != b	Έλεγχος ανισότητας τιμής
a <> b	Έλεγχος ανισότητας τιμής
a !== b	Έλεγχος ανισότητα τιμής και τύπου δεδομένων
a < b	Έλεγχος μικρότερης τιμής
a > b	Έλεγχος μεγαλύτερης τιμής
a <= b	Έλεγχος μικρότερης-ίσης τιμής
a >= b	Έλεγχος μεγαλύτερης-ίσης τιμής

(π.χ. "100"==100 →true
"100"===100→false (πιο αυστηρός έλεγχος))
=== είναι απλούστερος και πιο γρήγορος τελεστής.

11.9.4 The Strict Equals Operator (===)
The production
EqualityExpression : *EqualityExpression* === *RelationalExpression*
is evaluated as follows:
1.Let *lref* be the result of evaluating *EqualityExpression*.
2.Let *lval* be GetValue(*lref*).
3.Let *rref* be the result of evaluating *RelationalExpression*.
4.Let *rval* be GetValue(*rref*).
5.Return the result of performing the strict equality comparison *rval* === *lval*. (See 11.9.6)

11.9.3 The Abstract Equality Comparison Algorithm
The comparison *x* == *y*, where *x* and *y* are values, produces **true** or **false**. Such a comparison is performed as follows:
1.If Type(*x*) is the same as Type(*y*), then
1. If Type(*x*) is Undefined, return **true**.
2. If Type(*x*) is Null, return **true**.
3. If Type(*x*) is Number, then
1. If *x* is **NaN**, return **false**.
2. If *y* is **NaN**, return **false**.
3. If *x* is the same Number value as *y*, return **true**.
4. If *x* is **+0** and *y* is **−0**, return **true**.
5. If *x* is **−0** and *y* is **+0**, return **true**.
6. Return **false**.
4. If Type(*x*) is String, then return **true** if *x* and *y* are exactly the same sequence of characters (same length and same characters in corresponding positions). Otherwise, return **false**.
5. If Type(*x*) is Boolean, return **true** if *x* and *y* are both **true** or both **false**. Otherwise, return **false**.
6. Return **true** if *x* and *y* refer to the same object. Otherwise, return **false**.
2.If *x* is **null** and *y* is **undefined**, return **true**.
3.If *x* is **undefined** and *y* is **null**, return **true**.
4.If Type(*x*) is Number and Type(*y*) is String,
return the result of the comparison *x* == ToNumber(*y*).
5.If Type(*x*) is String and Type(*y*) is Number,
return the result of the comparison ToNumber(*x*) == *y*.
6.If Type(*x*) is Boolean, return the result of the comparison ToNumber(*x*) == *y*.
7.If Type(*y*) is Boolean, return the result of the comparison *x* == ToNumber(*y*).
8.If Type(*x*) is either String or Number and Type(*y*) is Object,
return the result of the comparison *x* == ToPrimitive(*y*).
9.If Type(*x*) is Object and Type(*y*) is either String or Number,
return the result of the comparison ToPrimitive(*x*) == *y*.
10.Return **false**.

Σχόλιο γραμμής
//αυτό είναι ένα σχόλιο
Σχόλιο πολλών γραμμών
/*σχόλιο
πολλαπλών γραμμών*/

ΤΕΛΕΣΤΗΣ typeof
Ο ειδικός αυτός τελεστής χρησιμοποιείται ως πρόθεμα ενός τελεστέου (μιας μεταβλητής) και επιστρέφει τον τύπο αυτού ως μια συμβολοσειρά.

Τύπος μεταβλητής	αποτέλεσμα
Undefined	"undefined"
Null	"object"
Boolean	"boolean"
Number	"number"
String	"string"
Symbol (νέο στην ECMAScript 2015)	"symbol"
Function object	"function"
Οποιοδήποτε άλλο αντικείμενο	"object"

if (typeof total=="number"){
…}

ΕΙΔΙΚΕΣ ΤΙΜΕΣ Infinity και NaN
Αριθμητικές ή άλλες πράξεις μπορούν να πράγουν κάποια από τις παρακάτω ειδικές τιμές:
Infinity Θετικό άπειρο π.χ. 100/0
-Infinity αρνητικό άπειρο
NaN Not-A-Number π.χ. 10/"abc"
Οι τύποι αυτοί μπορούν να εντοπιστούν με ειδικές συναρτήσεις π.χ. **x.isNaN()**

ΟΡΙΣΜΟΣ ΜΕΤΑΒΛΗΤΩΝ (let / var / const)
ES5:μόνο var
var myName='Nikos';//επιτρέπει πολλαπλές δηλώσεις π.χ. var myName='Chris';
var myAge=75;
ES6:δύο νέα keywords:let και const
let myName='Nikos';//Δεν επιτρέπει πολλαπλές δηλώσεις π.χ. var myName='Chris'; θα δώσει error
let myAge=75;
const pi=3.14; // η const **δεν επιτρέπει** πολλαπλές δηλώσεις **ούτε** αλλαγή τιμής της μεταβλητής
η const **ορίζει σταθερές** π.χ. const hoursInDay=24;
<u>Λόγοι χρήσης της const :</u>
-Ασφάλεια(εξασφαλίζουμε ότι κάποιος τρίτος δεν θα μπορεί να αλλάξει την τιμή)
-για εκσφαλμάτωση (ελέγχουμε πότε αλλάζει τιμή η μεταβλητή)

διαφορές let(ES6) και var(ES5)
-Διαφορές στη δυνατότητα επαναδήλωσης μιας μεταβλητής (στη var επιτρέπεται,στη let όχι)
-Διαφορές στην εμβέλεια (η var έχει εμβέλεια και έξω από το μπλοκ στο οποίο δηλώνεται)
η let ορίζει εμβέλεια της μεταβλητής μόνο στο μπλοκ στο οποίο έχει δηλωθεί(μπλοκ:μία εντολή if,for κλπ) ενώ η var έχει εμβέλεια σε ολόκληρο τον κώδικα π.χ.

```
for (var i=0;i<10;i++){ console.log(i) }
console.log(i)//θα εμφανίσει την τιμή του i
for (let j=0;j<10;j++){console.log(j) }
console.log(j)//θα δώσει error
```

-Διαφορές στην δυνατότητα πρόσβασης στην μεταβλητή πριν τη δήλωση της (στη var επιτρέπεται, η τιμή είναι αρχικά undefined, στη let όχι)

```
<script>
  console.log('x=', typeof x, x); // x undefined NaN
  console.log('y=', typeof y, y); // ReferenceError:
  Cannot access 'y' before initialization
  var x=5;
  let y=10;
</script>
```

-Γενικά:**προτιμούμε τη let με τους περιορισμούς της**

ΣΥΜΒΟΛΟΣΕΙΡΕΣ

Οριοθετούμε συμβολοσειρές με απλά ή διπλά εισαγωγικά ή δασεία.

```
let sql='Απλά εισαγωγικά.';
let dql="Διπλά εισαγωγικά.";
let quot=`Δασεία.`;
```

Χρησιμοποιούμε το ένα για να οριοθετήσουμε συμβολοσειρά που περιέχει το άλλο

```
let dblSgl="I'm feeling sad.";
```

χαρακτήρας διαφυγής \

```
let traveller='i\'ve been to India';
```

Σύνθεση(ένωση) συμβολοσειρών

```
let one='Hello';
let two='world!';
let join=one+two;
```

ΜΕΤΑΤΡΟΠΕΣ ΑΠΟ STRING ΣΕ NUMBER

```
//μετατροπή από string σε αριθμό
let myString='123';
let myNum=Number(myString);
typeof myNum;
```

```
//μετατροπή από αριθμό σε string
let myNum=123;
let myString=myNum.toString();
```

ΜΕΘΟΔΟΙ

```
let myName='Nikos';
```

ΜΕΘΟΔΟΣ	ΕΝΤΟΛΗ	ΑΠΟΤΕΛΕΣΜΑ
μήκος συμβολοσειράς	myName.length;	5
πρώτος χαρακτήρας	myName[0];	"N"
τελευταίος χαρακτήρας	myName[myName.length-1];	"s"
τμήμα από 0 έως 3	myName.slice(0,3);	'Nik'
θέση substring	myName.indexOf('kos');	2 (-1 αν δεν βρεθεί)
slice από 3 ως το τέλος	myName.slice(3);	'os'
μετατροπή σε πεζά	myName.toLowerCase();	"nikos"
μετατροπή σε κεφαλαία	myName.toUpperCase();	"NIKOS"
αντικατάσταση substring	myName.replace('kos','na')	"Nina"

template literals(από την έκδοση ES6)

Συμβολοσειρές που περικλείονται από την δασεία (`) περιέχουν αναφορά σε μεταβλητές με ${v}

π.χ.

```
let a=5;
let b=10;
console.log(`a+b=${a+b} ενώ 2a+b=${2*a+b}.`);
```

ενναλακτικά μπορεί να γραφεί:

```
console.log('a+b='+(a+b)+'ενώ 2a+b='+(2*a+b)+'.');
```

ΠΙΝΑΚΕΣ-arrays

οι πίνακες είναι αντικείμενα που περιέχουν πολλάπλες τιμές (όχι απαραίτητα του ίδιου τύπου) αποθηκευμένες ως ακολουθία

```
let shopping=['ψωμί','γάλα','τυρί','μήλα'];
let sequencee=[1,1,2,3,5,8,13];
let random=['tree',795,[0,1,2]];
```

αναφορά σε στοιχεία πίνακα,μήκος πίνακα

```
let seq=[1,1,2,3,5,8,13]
seq[2]//δείκτης σε στοιχείο πίνακα.Η θέση του στοιχείου μετράει από το 0.Άρα η θέση 2 είναι το 3ο στοιχείο του πίνακα με τιμή 2
seq[1]=8;//εκχώρηση τιμή 8 στην θέση 1 του πίνακα (→[1,8,2,3,5,8,13])
seq.length;//το πλήθος των στοιχείων του πίνακα(→7)
```

ΔΗΜΙΟΥΡΓΙΑ ΠΙΝΑΚΑ ΜΕΣΩ literal ή new Array()

Δύο ισοδύναμοι τρόποι μέσω literal ή με κλήση της συνάρτησης δημιουργού αντικειμένου Array()

```
let c=[10,20];
let c=new Array(10,20);//array με 2 στοιχεία
```

new Constructor-function:τελεστής δημιουργίας αντικειμένου μέσω συνάρτησης-δημιουργού(Array() είναι η συνάρτηση -δημιουργός πινάκων)

```
let c=new Array();//πίνακας μη ορισμένου μεγέθους
let c=new Array(2);//δημιουργεί το c=[undefined,undefined]
let f=new Array("5");//δημιουργεί έναν πίνακα με ένα στοιχείο το "5"
```

ΑΡΧΙΚΟΠΟΙΗΣΗ ΜΕ ΒΡΟΧΟ ΕΠΑΝΑΛΗΨΗΣ

```
//δημιουργία 2 νέων πινάκων
let n1=new Array(5);//allocate 5-element Array
let n2=new Array();//allocate empty Array

//αρχικοποίηση πίνακα με γνωστό μέγεθος
for (let i=0;i<n1.length;++i)
        n1[i]=i;
//αρχικοποίηση πίνακα χωρίς καθορισμένο μέγεθος
for (i=0;i<5;++i)
        n2[i]=i;
```

n.length το πλήθος των στοιχείων του πίνακα

ΜΕΘΟΔΟΙ ΠΙΝΑΚΩΝ

Προσθήκη/αφαίρεση στοιχείων σε πίνακα

-Η μέθοδος **pop** καταργεί το **τελευταίο στοιχείο του πίνακα**.Το στοιχείο που καταργήθηκε επιστρέφεται

-Η μέθοδος **push** προσθέτει ένα νέο στοιχείο **στο τέλος του πίνακα**.Το νέο μήκος του πίνακα επιστρέφεται.

π.χ.

```
let myArray=["Athens","Patras","Thessaloniki","Volos"]
myArray.push('Larissa');//επιστρέφει 5
myArray.push('Katerini','Kavala');//επιστρέφει 7
myArray.pop();//επιστρέφει "Kavala"
```

-Η μέθοδος **shift()** αφαιρεί ένα στοιχείο **από την αρχή του πίνακα**.Το στοιχείο του πίνακα επιστρέφεται.

-Η μέθοδος **unshift**(στοιχείο) προσθέτει ένα νέο στοιχείο **στην αρχή του πίνακα**.Το νέο μήκος του πίνακα επιστρέφεται.

```
let myArray=["Athens","Patras","Thessaloniki","Volos"]
myArray.unshift('Larissa');//επιστρέφει 5
myArray.unshift('Katerini','Kavala');//επιστρέφει 7
myArray.shift();//επιστρέφει "Katerini"
```

από string σε array split() από array σε string join()

```
let myData = 'Athens, Patras, Thessaloniki, Volos';
let myArray = myData.split(','); //
(4) ["Athens", "Patras", "Thessaloniki", "Volos"]
    0: "Athens"
    1: "Patras"
    2: "Thessaloniki"
    3: "Volos"
let myNewString = myArray.join(' ');
myNewString
"Athens Patras Thessaloniki Volos"
```

ΤΕΛΕΣΤΗΣ const ΣΕ ΟΡΙΣΜΟ ΠΙΝΑΚΑ

```
const a=[]; //ένα αντικείμενο μπορεί να οριστεί με τον τελεστή const,
            // όμως το περιεχόμενο του μπορεί να τροποποιηθεί
a.push(1);
a.push(5,6);
a
[1,5,6]
```

ΕΙΣΑΓΩΓΗ ΚΑΙ ΔΙΑΓΡΑΦΗ ΣΤΟΙΧΕΙΩΝ ΑΠΟ array

splice(θέση,πλήθος,στοιχεία)

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

θέση εισαγωγής/διαγραφής

αριθμός στοιχείων προς διαγραφή

στοιχεία προς εισαγωγή

```
αποτέλεσμα:
Banana,Orange,Lemon,Kiwi,Apple,Mango
```

ΠΡΟΣΘΗΚΗ ΣΤΟΙΧΕΙΟΥ ΣΕ array

```
let a=[1,"10",3.14];
a[10]='kostas';
```

βάζει στην θέση 10 το 'kostas' και **γεμίζει τις θέσεις που δεν υπήρχαν** πριν με **undefined**

[1,"10",3.14,undefined,undefined,undefined,undefined,undefined,undefined,undefined,"kostas"]

ΤΑΞΙΝΟΜΗΣΗ ΠΙΝΑΚΑ

```
let fruits=["Banana","Orange","Apple","Mango"];
fruits.sort();
```

-ταξινομεί τα στοιχεία **αλφαβητικά**,μπορούσε όμως να περάσουμε ως όρισμα της sort() μια συνάρτηση ταξινόμησης που ορίζει τη σειρά **ανάλογα** με την τιμή που **επιστρέφει**

```
let points=[40,100,1,5,25,10];
points.sort(function(a,b){return a-b});// [1, 5, 10, 25, 40, 100]
```

ΕΝΤΟΛΕΣ ΥΠΟ-ΣΥΝΘΗΚΗ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ

-**if** (συνθήκη){//αν η συνθήκη είναι αληθής τότε
    εντολές
} else if (συνθήκη2) {//αλλιώς αν συνθήκη2 είναι αληθής τότε
    εντολές
} else {//αλλιώς
    εντολές
};

π.χ.

```
if (choice==='sunny'){
        para.textContent='Πάμε για μπάνιο.';
else if (choice==='rainy'){
        para.textContent='Πάρε ομπρέλα';}
else if (choice==='snowing'){
        para.textContent='Να πάμε για σκι';}
else{para.textContent='';
}
```

-**(συνθηκη)?run if true:run if false;**

π.χ.

```
let greeting=(isBirthday)?'Happy birthday!':'Good morning';
```

-switch(έκφραση/τιμή) {//παίρνει μία εκφραση/τιμή και ε**λέγχει ένα πλήθος επιλογών** μέχρι να βρει αυτή που ταιριάζει στην τιμή, οπότε **εκτελεί τον αντίστοιχο κώδικα**

```
        case choice1:
                εντολές
                break;
        case choice2:
                εντολές
                break;
        default: εντολές
}
```

π.χ.

```
switch(choice){
        case 'sunny':
                para.textContent='Πάμε για μπάνιο.';
                break;
        case 'rainy':
                para.textContent='Πάρε ομπρέλα';
                break;
        case 'snowing':
                para.textContent='Να πάμε για σκι';
                break;
        default:
                para.textContent='';
                break;  }
```

-**Διαχείριση σφαλμάτων**

try{ εντολές
}catch(err){
εντολές
}

```
<script>
  try {
    adddlert("Welcome guest!");
  }
  catch(err) {
    document.querySelector("#demo").innerHTML = "ERROR: "
  + err.message;
  }
</script>
```

ΔΟΜΗ ΕΠΑΝΑΛΗΨΗΣ for

for(αρχικοποίηση**;**συνθήκη τερματισμού**;**βήμα αύξησης){//κώδικας}

```
const friends = ['Nikos', 'Kostas', 'Andreas'];
let info = 'My friends are :';
const para = document.querySelector('p');
for (let i = 0; i < friends.length; i++) {
      info += friends[i] + ', '; }
para.textContent = info;
```

```
for (let i = 0; i < friends.length; i++) {
    if (i === friends.length - 1) {
        info += 'and ' + friends[i] + '.'; }
    else {
        info += friends[i] + ', '; }
}
```

Άσκηση

Να κατασκευάσετε συνάρτηση που παράγει ιστοσελίδα με το εξής περιεχόμενο

HTML font size 5ex

HTML font size 6ex

HTML font size 7ex

```
<script>
    function loadContent(){
        b = document.querySelector("body");
        for (let counter=1; counter <= 7; counter++) {
            let para = document.createElement("p"); // create a <p> node
            para.style.fontSize = counter + "em"; // append the text to <p>
            let t = document.createTextNode("HTML font size " + counter + "ex"); //
            append the text to <p>
            para.appendChild(t); // append the text to <p>
            b.appendChild(para); // append the <p> to <body>
        }
    }
    document.onload = loadContent();
</script>
```

for(item **of** object){//κώδικας}

```
let i = 0;
for (friend of friends) {
    if (friends.indexOf(friend) === friends.length-1 ) {
        info += 'and ' + friend + '.'; }
    else {
        info += friend + ', ';
        }
    i++;
}
```

```
for (item of object) {
    κώδικας
}
```

for(key **in** array){//κώδικας}

-Χρησιμοποιείται με μονοδιάστατους ή πολυδιάστατους πίνακες

-Η for …in, σαρώνει διαδοχικά, όλα τα στοιχεία του βασικού επιπέδου του πίνακα από την αρχή έως το τέλος του , τόσες επαναλήψεις, όσες και το πλήθος των στοιχείων της διάστασης του πίνακα.

-Σε κάθε επανάληψη, το κλειδί (δείκτης) του επόμενου στοιχείου
του πίνακα αντιγράφεται σε μια μεταβλητή προς χρήση εντός του βρόχου.

π.χ.

```
arrayVar=[10,20,40,50,60] //υπολογισμός αθροίσματος στοιχείων του πίνακα
let sum=0;
for (let key in arrayVar){
        sum+=arrayVar[key];
}
console.log(sum);
```

ΔΟΜΗ ΕΠΑΝΑΛΗΨΗΣ WHILE

```
let i = 0;
while (i < friends.length) {
      if (i === friends.length - 1) {
            info += 'and ' + friends[i] + '.'; }
      else {
            info += friends[i] + ', ';
            }
      i++;
}
```

```
while(condition) {
      κώδικας
}
```

DO - WHILE(εκτελείται τουλάχιστον 1 φορά**)**

```
let i = 0;
do {
      if (i === friends.length - 1) {
            info += 'and ' + friends[i] + '.'; }
      else {
            info += friends[i] + ', ';
            }
      i++;
} while (i < friends.length)
```

```
do {
      κώδικας
} while( condition );
```

ΒΡΟΧΟΙ ΕΠΑΝΑΛΗΨΗΣ ΜΕ break

Η εντολή break τερματίζει βρόχους while,do-while,for, ή switch μέσα στους οποίους βρίσκεται και μεταφέρει έλεγχο προγράμματος στην επόμενη εντολή

```
for (let i = 0; i < a.length; i++) {
      if (a[i] == theValue) {
            break;
      }
}
```

διαπέραση των στοιχείων ενός array μέχρι να βρεθεί ο δείκτης του στοιχείου με τιμή theValue

ΒΡΟΧΟΙ ΕΠΑΝΑΛΗΨΗΣ ΜΕ continue

Η εντολή **continue** μέσα σε ένα βρόχο while,do-while,for επαναφέρει έλεγχο **στην αρχή του βρόχου.**

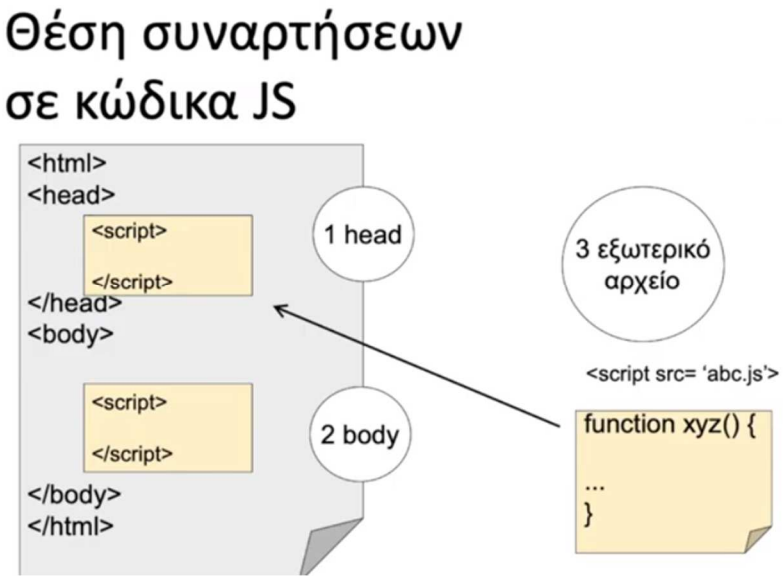
```
let i = 0;
let n = 0;
while (i < 5) {
      i++;
      if (i == 3) {
            continue;
      }
      n += i;
      console.log(n); }
```

Εκτύπωση ενδιάμεσου αθροίσματος των αριθμών από1 έως 5 αγνοώντας τον αριθμό 3
δηλαδή τυπώνει 1 3 7 12

```
1=1
1+2=3
1+2+4=7
1+2+4+5=12
```

ΣΥΝΑΡΤΗΣΕΙΣ

-Η **JavaScript** είναι ισχυρά συναρτησιακή γλώσσα, χειρίζεται τις συναρτήσεις ως αντικείμενα **(callable objects)**
-Το **όνομα μιας συνάρτησης** θα πρέπει απαραίτητα να ξεκινάει από ένα αλφαβητικό χαρακτήρα (κεφαλαία ή πεζά) ή το χαρακτήρα της κάτω παύλας (underscore)
-Εξ ορισμού, οι τιμές των **ορισμάτων εισόδου** αποδίδονται στον κώδικα της συνάρτησης με αντιγραφή τους σε τοπικές μεταβλητές (**κλήση μέσω τιμής— by value**)
-Αν το όρισμα εισόδου είναι κάποιος **πίνακας** ή άλλο **αντικείμενο,**αυτό αποδίδεται στην εσωτερική τοπική μεταβλητή μέσω αναφοράς (**by reference**)



-Συνάρτηση=επαναχρησιμοποιούμενο μπλοκ κώδικα
-Οι συναρτήσεις που ορίζονται **εντός** αντικειμένων objects ονομάζονται **μέθοδοι(methods)**

```
function fname (params) {
  εντολές
  return values;
}
```

```
function random(number) {
  // τυχαίος αριθμός μεταξύ 1 και number
  return Math.floor(Math.random()*number+1);
}

let x = random(5);  //κλήση συνάρτησης
```

ΕΝΑΛΑΚΤΙΚΟΣ ΤΡΟΠΟΣ ΝΑ ΚΑΛΕΣΩ ΜΙΑ ΣΥΝΑΡΤΗΣΗ(literal notation)

Function literal notation:εναλλακτικός τρόπος σύνταξης συνάρτησης-οφείλει την ύπαρξη της στο γεγονός ότι η Javascript αντιμετωπίζει και τις συναρτήσεις ως αντικείμενα.
Μετά την δήλωση συνάρτησης,αυτή μπορεί να κληθεί βάσει του ονόματος της.

```
let όνομα_συνάρτησης =
 function( ορίσματα ) {
    //κώδικας
    return δεδομένα;
}
```

ΑΝΩΝΥΜΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Οι ανώνυμες συναρτήσεις χρησιμοποιούνται μαζί με ένα event handler.
παράδειγμα ένα button που καλεί alert('hello')

```
let myButton = document.querySelector('button');

myButton.onclick = function() {
      alert('hello');
}
```

```
function (params) {
  statement(s);
  return values;
}
```

ES6 arrow functions

x= (name) => {console.log('hi '+name);}
//name ορίσματα της συνάρτησης
//μετά το => οι εντολές της συνάτησης

π.χ. x('Nikos')
hi Nikos

-αν η συνάρτηση περιέχει μόνο μια εντολή return απλοποιείται σε (val)=>έκφραση
π.χ. let x=(z)=>z**2;//z στο τετράγωνο
x(5)
25

Η λέξη this

Η λέξη **this** παίρνει διαφορετικές τιμές ανάλογα με το πού χρησιμοποιείται:

- Σε μια μέθοδο, αναφέρεται στο αντικείμενο ιδιοκτήτη της μεθόδου.
- Σε μια συνάρτηση, αναφέρεται στο αντικείμενο global.
- Σε μια συνάρτηση, σε strict mode, είναι undefined.
- Σε event, αναφέρεται στο στοιχείο που σχετίζεται με το event.

π.χ.

```
<!DOCTYPE html>
<html>
    <head>
        <style>
           button{
               background-color: grey;
               width:300px;
               height:100px;
               font-size:20px;
               text-decoration: none;
           }
           p{font-size:20px;font-weight: bold;}
        </style>
    </head>
    <body>
        <p>Παράδειγμα της λέξης this</p>
        <script>
            function myfunc(element){
                element.style.backgroundColor='yellow';
            }
        </script>
        <button onclick="myfunc(this)">αλλαγή χρώματος</button>
    </body>
</html>
```

ΕΜΒΕΛΕΙΑ ΜΕΤΑΒΛΗΤΩΝ

-Έξω από όλες τις συναρτήσεις ορίζεται περιοχή καθολικής εμβέλειας **global scope**.
Μεταβλητές που ορίζονται στο επίπεδο αυτό είναι προσβάσιμες από όλον τον κώδικα.

-Κάθε συνάρτηση ορίζει τοπική εμβέλεια **local scope** και οι μεταβλητές που ορίζονται στο επίπεδο αυτό είναι προσβάσιμες **μόνο** μέσα στη συνάρτηση

1. Τοπικές μεταβλητές

```
function myFunction() {
   var userName = "Nikos";
   /* το userName μπορεί να
      χρησιμοποιηθεί εδώ */
   console.log(userName); // ok
}
/* Ο κώδικας δεν μπορεί να
χρησιμοποιήσει το userName */
console.log(userName); //error
```

ES6: `let userName = "Nikos";`
(μεταβλητή με εμβέλεια block)

2. Καθολικές μεταβλητές

```
let userName = "Maria";
// μπορεί να χρησιμοποιηθεί η userName
console.log(userName); // ok

function myFunction() {
  // μπορεί να χρησιμοποιηθεί η userName
  console.log(userName); // ok
}
```

ΑΥΤΟΜΑΤΑ ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ

Αν ανατεθεί τιμή σε μεταβλητή που δεν έχει δηλωθεί αυτή αυτόματα γίνεται καθολική μεταβλητή(δεν επιτρέπεται σε strict mode)

```
function myFunction(){
        username="Nikos";
        //μπορεί να χρησιμοποιηθεί η userName
}
myFunction()
//μπορεί να χρησιμοποιηθεί η userName
```

ΕΚΤΕΛΕΣΗ JS ΣΕ strict mode

"use strict";//ορίστηκε μετά την ES5
Πιο αυστηρή έκδοση της JS που δεν επιτρέπει κάποιες "χαλαρές" λειτουργίες,όπως η χρήση μεταβλητών που δεν έχουν δηλωθεί.

```
"use strict";
x=3.14;//σφάλμα γιατί η x δεν έχει δηλωθεί
```

ΠΡΟΑΙΡΕΤΙΚΑ ΟΡΙΣΜΑΤΑ ΣΥΝΑΡΤΗΣΕΩΝ

Η javascript μέχρι την έκδοση ES6 δεν υποστήριζε άμεσα προαιρετικά ορίσματα με εξ'ορισμού τιμές. Λύση ήταν ο εσωτερικός έλεγχος με επιπλέον κώδικα που εντοπίζει τη μη χρήση του ορίσματος και του αναθέτει μια τιμή.

```
function myOwnFunction( optional ) {
   if ( typeof optional == 'undefined' )
   optional = 1;
   return optional + 1;
}

myOwnFunction(3); // Επιστρέφεται 4
myOwnFunction(); // Επιστρέφεται 2
```

-στην έκδοση ES6 επιτρέπεται πλέον προαιρετικά ορίσματα με προκαθορισμένες τιμές

```
function myOwnFunction( optional=1 ) {
   return optional + 1;
}

myOwnFunction(3); // Επιστρέφεται 4
myOwnFunction(); // Επιστρέφεται 2
```

ΣΥΝΑΡΤΗΣΕΙΣ ΑΡΙΘΜΩΝ

`int parseInt( string data [, int radix ] )`	Εξαγωγή ενός ακέραιου αριθμού από συμβολοσειρά. Το προαιρετικό όρισμα εισόδου radix επιτρέπει τη σωστή αναγνώριση του αριθμού σε ένα από τα αριθμητικά συστήματα: δεκαδικό (10), δεκαεξαδικό (16), δυαδικό (2) ή οκταδικό (8).
`float parseFloat( string data )`	Εξαγωγή ενός πραγματικού αριθμού από συμβολοσειρά.
`bool isNaN( mixed numdata )`	Έλεγχος αν η μεταβλητή είναι (ακέραιος ή πραγματικός) αριθμός ή συμβολοσειρά που αναπαριστά αριθμό.
`bool isFinite( mixed numdata )`	Έλεγχος αν η μεταβλητή είναι έγκυρος, πεπερασμένος (Finite) ακέραιος ή πραγματικός αριθμός, ή συμβολοσειρά που αναπαριστά πεπερασμένο αριθμό.

ΜΕΘΟΔΟΙ ΔΙΑΠΕΡΑΣΗΣ ΠΙΝΑΚΩΝ

-myArray.**forEach**(myFunction) εφαρμόζει τη συνάρτηση σε κάθε στοιχείο του πίνακα
π.χ.
let myAr=["Χίος","Μυτιλήνη","Σάμος"]
myAr.forEach((el)=>console.log("Η "+el));
Η Χίος
Η Μυτιλήνη
Η Σάμος

-myArray.**map**(myFunction) δημιουργεί νέο πίνακα με εφαρμογή της συνάρτησης σε κάθε στοιχείο του
π.χ.
let myAr=["Χίος","Μυτιλήνη","Σάμος"]
let myAr=myAr.map((el)=>"Η νήσος "+el);

newAr
(3) ["Η νήσος Χίος","Η νήσος Μυτιλήνη","Η νήσος Σάμος"]

-myArray.**filter**(myFunction) δημιουργεί νέο πίνακα με τα στοιχεία που ικανοποιούν τη συνάρτηση φίλτρο
π.χ.
let myAr=["Χίος","Μυτιλήνη","Σάμος"]
let newAr=myAr.filter((el)=>el.length>5);

newAr
["Μυτιλήνη"]

-myArray.**reduce**(acumFun) παράγει μια τιμή μετά από διαδοχική εφαρμογή της συνάρτησης acumFun στα στοιχεία του πίνακα.Προσοχή η accumFun παίρνει ως ορίσματα(acum,val)
π.χ.
let myAr=["Χίος","Μυτιλήνη","Σάμος"]
let islands="";
let result=myAr.reduce((islands,el)=>islands+= " "+el);

result
"Χίος Μυτιλήνη Σάμος"

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

ΔΙΑΧΕΙΡΙΣΗ ΣΥΜΒΑΝΤΩΝ(events)

Events (συμβάντα) είναι **ενέργειες ή γεγονότα** που συμβαίνουν, για τα οποία το σύστημα μας ενημερώνει, ώστε να μπορούμε να ανταποκριθούμε σε αυτά :

- Ολοκληρώθηκε η φόρτωση της ιστοσελίδας.
- Ο χρήστης κάνει κλικ με το ποντίκι πάνω από ένα συγκεκριμένο στοιχείο
- Ο χρήστης τοποθετεί τον δρομέα πάνω από ένα συγκεκριμένο στοιχείο.
- Ο χρήστης πατάει ένα πλήκτρο στο πληκτρολόγιο.
- Ο χρήστης αλλάζει μέγεθος ή κλείνει το παράθυρο του προγράμματος περιήγησης.
- Ένα βίντεο που αναπαράγεται ή τίθεται σε παύση ή τελειώνει.
- Παρουσιάστηκε κάποιο σφάλμα

ΧΕΙΡΙΣΤΕΣ ΣΥΜΒΑΝΤΩΝ(event handlers)

-Ένα συμβάν μπορεί να έχει ένα χειριστή συμβάντος **event handler** που είναι ένα μπλοκ κώδικα (συνήθως μια συνάρτηση javascript) που θα τρέξει όταν προκύψει το συμβάν.

-Όταν ορίσουμε το μπλοκ κώδικα να τρέχει ως απάντηση σε συμβάν αναφερόμαστε σε **καταχώρηση ενός χειριστή σμβάντος.**

ΚΑΤΑΧΩΡΗΣΗ ΧΕΙΡΙΣΤΗ ΣΥΜΒΑΝΤΩΝ

1. Inline.

```
<button onclick="myfunc(this)">κλικ
</button>
```

2. Ιδιότητες αντικειμένων DOM on-event

```
window.onload = () => {
     //window loaded}
```

3. Με τον addEventListener()

```
window.addEventListener('load', () => {
     //window loaded
     })
```

π.χ.

inline

όταν επιλεγεί αυτό το κείμενο θα κληθεί η συνάρτηση message

ο τρόπος αυτός δεν συνιστάται αφού έτσι ανακατεύουμε κώδικα HTML , JAVASCRIPT

σαν ιδιότητα on

```
<button>Change color</button>
let btn=document.querySelector('button');
function rand(number){
        return Math.floor(Math.random()*(number+1));
}
btn.onclick=function(){
        let rndCol='rgb('+rand(255)+','+rand(255)+','+rand(255)+')';
document.body.style.backgroundColor=rndCol;}
```

Η ιδιότητα **onclick** δεν είναι ιδιότητα του <button> αλλά είναι ένας ειδικός τύπος

-όταν οριστεί ίσος με κάποιο κώδικα,ο κώδικας αυτός θα εκτελείται όταν το συμβάν προκύψει στο πλήκτρο.

addEventListener/removeEventListener

Χρησιμοποιούνται για δημιουργία και αναίρεση χειριστών συμβάντων

Σύνταξη : addEventListener(συμβάν,χειριστής)

πλεονέκτημα:ο event listener μπορεί να αναιρεθεί

π.χ. btn.addEventListener('click',bgChange);

ΚΑΤΗΓΟΡΙΕΣ ΣΥΜΒΑΝΤΩΝ
window.onload=f() ή document.onload=f()
Το document.onload=αφορά φόρτωμα του DOM
Το window.onload=περιλαμβάνει επίσης φόρτωμα των εικόνων και άλλων βοηθητικών αρχείων

```
document.onload=function(){
….
}
window.onload=function(){
….
}
```

ΣΥΝΗΘΗ ΣΥΜΒΑΝΤΑ
• **btn.onfocus** και **btn.onblur** —Συμβάντα που προκύπτουν όταν το btn έχει την εστίαση του δρομέα ή όχι
• **btn.ondblclick** --Συμβάν που προκύπτει όταν κάνουμε διπλό κλικ στο btn.
• **window.onkeypress**, **window.onkeydown**, **window.onkeyup** —Ένα πλήκτρο πιέζεται στο πληκτρολόγιο. Το **keypress** αναφέρεται σε ένα γενικό πάτημα (κουμπί κάτω και έπειτα επάνω), ενώ η **keydown** και το **keyup** αφορούν μόνο το πλήκτρο κάτω και το πλήκτρο επάνω αντίστοιχα.
• **btn.onmouseover** και **btn.onmouseout** Ο δείκτης μετακινείται ώστε να αιωρείται πάνω από το btn, ή όταν σταματά να αιωρείται πάνω από το κουμπί και να απομακρύνεται από αυτό, αντίστοιχα.

ΑΝΤΙΚΕΙΜΕΝΑ ΣΤΗΝ javascript
Τα **objects** είναι δομές που περιέχουν ζεύγη **ιδιότητας/τιμής**(property/value)
Οι ιδιότητες μπορεί να πάρουν ως τιμή σταθερές,συναρτήσεις (**μέθοδοι**) ή άλλα αντικείμενα
π.χ.

```
let car={//αντικείμενο car
        make:"volvo",
        speed:140,
        engine:{
                size:1800,
                fuel:"diesel",
                pistons:["piston1","piston2"]},
        drive:function(){return "drive…"}}
```

π.χ.

```
const person1={//αντικείμενο person1
firstName:"Γιάννης",
lastName:"Ιωάννου",
age:50,
eyeColor:"πράσινα",
greeting:function(){
        alert('Γεια! με λένε '+this.lastName+'.');}}
```

σημείωση: αν στην greeting είχαμε array function δεν θα δούλευε το this για το αντικείμενο

```
greeting:()=>(){alert('Γεια! με λένε '+this.lastName+'.');}
```

ΑΝΑΦΟΡΑ ΣΤΙΣ ΙΔΙΟΤΗΤΕΣ ΕΝΟΣ ΑΝΤΙΚΕΙΜΕΝΟΥ
Έστω ότι θέλουμε την ιδιότητα firstName του αντικειμένου person1
a)person1.firstName b)person["firstName"]
αντίστοιχα για μεθόδους
a)person.greeting() b)person1["greeting"]()

ΔΗΜΙΟΥΡΓΙΑ ΑΝΤΙΚΕΙΜΕΝΟΥ

παράδειγμα συνάρτησης κατασκευαστή -αντικειμένων που επιστρέφει το νέο αντικείμενο.

```
function createNewPerson(name){
	let obj={};
	obj.name=name;
	obj.greeting=function(){
		alert('Γεια με λένε '+this.name+'.');};
		return obj;}

let kostas=createNewPerson('Κώστα');
```

δημιουργία με τη λέξη new

```
function Person(name){
	this.name=name;
	this.greeting=function(){
		alert('Γεια με λένε "+this.name+'.');
	};
}
let p1=new Person('Bob');//η κλήση της new Person() δημιουργεί αντικείμενα
```

δημιουργία αντικειμένου με κλάσεις(νέο στην ES6)

```
class Person{
	constructor(name){//δημιουργός αντικειμένου
		this.name=name;}
	greeting(){//μόνο έτσι ορίζονται μέθοδοι για να μπορούν να κληρονομηθούν
		return 'Γεια,είμαι ο '+this.name+'.';}
}
let p1=new Person('Κώστας');
```

ΙΔΙΟΤΗΤΑ __proto__

Τα αντικείμενα που δημιουργούνται με τους τρόπους που είδαμε περιέχουν το αντικείμενο **__proto__** που αφορά **το prototype,**μηχανισμό της JS για κληρονομιά ιδιοτήτων από το "object"

```
> p1
Person {name: "Κώστας"}
   name: "Κώστας"
   __proto__:
      constructor: class Person
      greeting: f greeting()
      __proto__: Object
```

ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

```
class Player extends Person{
	greeting(){
		return super.greeting()+',παίκτης'}
}
const messi=new Player('Messi');
messi.greeting();
```

επιστρέφει-→"Γεια σα είμαι ο Messi,ο παίκτης"

ΣΥΝΑΡΤΗΣΕΙΣ get set

```
class Person{
        get fullName(){
                return `${this.firstName} ${this.lastName}`;}
}
class Person{
        set age(years){
                this.theAge=years;}
}
```

ΜΕΘΟΔΟΙ ΤΟΥ MATH

```
Math.abs(x), Math.ceil(x), Math.cos(x)
Math.exp(x), Math.floor(x), Math.log(x),
Math.max(a,b), Math.min(a,b),
Math.pow(x,y), Math.round(x), Math.sin(x),
Math.sqrt(x), Math.tan(x) Math.PI, Math.E
```

```
> Math.abs(-250)
<- 250
```
-απόλυτη τιμή

```
> Math.ceil(3.14)
<- 4
```
-στρογγυλοποίηση προς τον επόμενο ακέραιο

```
> Math.floor(3.14)
<- 3
```
-στρογγυλοποίηση προς τον προηγούμενο ακέραιο

```
> Math.round(3.14)
<- 3
```
-στρογγυλοποίηση

```
> Math.round(3.7)
<- 4
```
-στρογγυλοποίηση

εύρεση του μεγίστου 3 αριθμών

```
> let maximum = (x,y,z) => {
      return Math.max(Math.max(x,y), z);}
<- undefined
> maximum(10,20,30)
<- 30
```

Τυχαίος αριθμός μεταξύ 0-1

```
> Math.random()
<- 0.5630060894081765
```

παράδειγμα με ζάρι(1 έως 6)

```
> let dice = () => { return Math.floor(Math.random()*6 +1);}
<- undefined
```

ΤΟ ΑΝΤΙΚΕΙΜΕΝΟ DATE

let newDate=new Date();

getFullYear()	Ανάκτηση του **έτους** ως τετραψήφιου αριθμού (yyyy)
getMonth()	Ανάκτηση του **μήνα** ως αριθμού (0-11)
getDate()	Ανάκτηση της **ημέρας** ως αριθμού (1-31)
getHours()	Ανάκτηση της **ώρας** (0-23)
getMinutes()	Ανάκτηση των **λεπτών** (0-59)
getSeconds()	Ανάκτηση **δευτερόλεπτων** (0-59)
getMilliseconds()	Ανάκτηση **millisecond** (0-999)
getTime()	Ανάκτηση του χρόνου (milliseconds από 1, Ιανουαρίου 1970)
getDay()	Ανάκτηση της ημέρας ως αριθμού (0-6)
Date.now()	Ανάκτηση του χρόνου. ECMAScript 5.

π.χ.

```
let dateString="Σήμερα είναι: ";
let newDate=new Date();//get month,day,and year.
dateString+=newDate.getDate()+"/";
dateString+=(newDate.getMonth()+1)+"/";
dateString+=newDate.getFullYear();
console.log(dateString);
```

CLOSURES

κάθε συνάρτηση όταν καλείται εκτελείται στο πλαίσιο στο οποίο ορίστηκε όχι στο πλαίσιο στο οποίο γίνεται η κλήση της, άρα μπορεί να αναφερθεί στις μεταβλητές που είχαν οριστεί σε εκείνο το πλαίσιο.

Συναρτήσεις που δημιουργούνται μέσα σε συναρτήσεις και καλούνται διατηρούν την κατάσταση τους μεταξύ διαδοχικών κλήσεων,λόγω αναφοράς στη **lexical closure "Enviroment"**

π.χ.

```
function makeCounter(){
        let count=0;
        return ()=>++count;//έχει πρόσβαση στην count
}
let counter=makeCounter();
for (let _i=0l_i<10;_i++){
        console.log(counter());}
```

```
const counter = function() {
  let privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;}
  return {
    increment: function() {
      changeBy(1);},
    decrement: function() {
      changeBy(-1);},
    value: function() {
      return privateCounter;}
  };
}();
```

```
console.log(counter.value()); // logs 0
counter.increment();
counter.increment();
console.log(counter.value()); // logs 2
counter.decrement();
console.log(counter.value()); // logs 1
```