

# JavaScript

## ΜΑΘΗΜΑ 9.3

### REGULAR EXPRESSIONS

#### ΠΕΡΙΕΧΟΜΕΝΑ:

1. Regular Expressions
  1. Μέθοδοι Συμβολοσειρών για regexp
  2. Μέθοδοι του RegExp.prototype
2. Συντακτικό RE:
  1. Στοιχειώδη Ταιριάσματα
  2. Επανάληψη
  3. Αποφυγή - Ειδικοί Χαρακτήρες
  4. OR και Group Χαρακτήρων
  5. Ομάδες Χαρακτήρων
  6. Ταίριασμα στα Άκρα

Βασιλική Κ.

Χρυσός Χορηγός Μαθήματος

## ΜΑΘΗΜΑ 9.3: Regular Expressions

### 1. Regular Expressions

Τα **regular expressions** (regex, re, μτφ: κανονικές εκφράσεις) είναι μία γλώσσα περιγραφής ομάδων συμβολοσειρών.

- π.χ. η κανονική έκφραση **ab.\*cd** περιγράφει τις συμβολοσειρές που ξεκινούν με ab και τελειώνουν με cd (το .\* == οτιδήποτε)
- και λέμε ότι π.χ. η συμβολοσειρά «abacd» **ταιριάζει (matches)** στην κανονική έκφραση (μπορεί να προκύψει από αυτήν)
- ενώ π.χ. η συμβολοσειρά «...» δεν ταιριάζει στην καν.έκφραση.

#### Κατασκευή Κανονικής Έκφρασης:

- Η κανονική έκφραση είναι ένα αντικείμενο που ορίζεται με δύο τρόπους:
  - **Literal:** Ορίζεται μέσα σε slashes, π.χ. η παραπάνω κανονική έκφραση ορίζεται ως: **/ab.\*cd/**
  - **Μέσω Συνάρτησης - Κατασκευαστή** (RegExp), π.χ. η παραπάνω κανονική έκφραση ορίζεται ως: **new RegExp("ab.\*cd")**

#### Υπενθύμιση από μάθημα 9.2:

- Η μέθοδος search() των συμβολοσειρών δουλεύει με κανονικές εκφράσεις:

Μέθοδος	Επεξήγηση
search(re)	Θέση 1 <sup>ης</sup> εμφάνιση ταιριάσματος της re στην συμβολοσειρά

#### Παράδειγμα 1: search

```
let text = "Computer Science is no more about computers " +
    "than astronomy is about telescope";
let regexp = /computer/
let regexp2 = new RegExp('science');
console.log(text.search(regexp));
console.log(text.search(regexp2));
```

#### Flags κανονικών εκφράσεων

- Μέσω flags μπορούμε να θέσουμε παραμέτρους στην κανονική έκφραση.
- Για παράδειγμα το flag 'i', κάνει case-insensitive αναζήτηση.
- Τα flags συντάσσονται κατά την κατασκευή των regex ως:
  - **Literal:** π.χ. **/regexp/i**
  - **Μέσω Κατ/στή:** π.χ. **new RegExp("regexp", "i")**

#### Σημείωση:

- Υπάρχουν κι άλλα flags (θα τα δούμε στις επόμενες διαφάνειες)

#### Παράδειγμα 2: search\_insensitive

```
let text = "Computer Science is no more about computers " +
    "than astronomy is about telescope";
let regexp = /computer/i
let regexp2 = new RegExp('science', 'i');
console.log(text.search(regexp));
console.log(text.search(regexp2));
```

- Η μέθοδος `match()` επιστρέφει όλα τα ταιριάσματα της κανονικής έκφρασης επί της συμβολοσειράς:

Μέθοδος	Επεξήγηση
<code>match(re)</code>	Πίνακας με τα ταιριάσματα της κ.ε. <code>re</code>

- Προσοχή ότι η συμπεριφορά της εξαρτάται από την ύπαρξη του flag 'g' (=global)
  - Αν το flag χρησιμοποιηθεί, τότε πράγματι επιστρέφει πίνακα με όλα τα ταιριάσματα
  - Αν το flag δεν χρησιμοποιηθεί, τότε επιστρέφει μόνο το πρώτο ταιρίασμα (μέσω ενός αντικειμένου που περιέχει επιπλέον πληροφορίες - groups (επόμε. διαφάνεια))

### Παράδειγμα 3: match

```
let text = "Computer Science is no more about computers " +
    "than astronomy is about telescope";
```

```
let regexp = /computer/gi
```

```
let regexp2 = /computer/i
```

```
console.log(text.match(regexp));
```

```
console.log(text.match(regexp2));
```

```
▼ Array(2)
  0: "Computer"
  1: "computer"
  length: 2
  ▶ [[Prototype]]: Array(0)
```

```
▼ Array(1)
  0: "Computer"
  groups: undefined
  index: 0
  input: "Computer Science is no more about compu
  length: 1
  ▶ [[Prototype]]: Array(0)
```

### Άλλες Μέθοδοι Συμβολοσειρών που υποστηρίζουν κανονικές εκφράσεις:

Μέθοδος	Επεξήγηση
<code>replace(regexp, replacement)</code>	Αντικαθιστά την πρώτη εμφάνιση της <code>regexp</code> με τη συμβολοσειρά (ή συνάρτηση) <code>replacement</code> [Με το flag <code>/g</code> κάνει όλες τις αντικαταστάσεις]
<code>split(regexp[, n])</code>	Χωρίζει τη συμβολοσειρά σε μέρη, χρησιμοποιώντας την <code>regexp</code> ως διαχωριστή. Τα αποτελέσματα επιστρέφονται σε πίνακα [Αν οριστεί το <code>n</code> , τότε θα επιστρέψει πίνακα με τα πρώτα <code>n</code> αποτελέσματα]
<code>matchAll(regexp)</code>	Ίδια με τη <code>match()</code> [ωστόσο απαιτεί το <code>/g</code> , επιστρέφει iterator]
<code>replaceAll(re, repl)</code>	Ίδια με τη <code>replace()</code> [απαιτεί το <code>/g</code> ]

### Παράδειγμα 4: string methods

```
console.log(text.split(/ /));
console.log(text.split(/ /, 3));
console.log(text.split(/computer/i));
```

```
console.log(text.replace(/computer/i, "math"));
console.log(text.replace(/computer/ig, "math"));
console.log(text.replaceAll(/computer/ig, "math"));
```

```
console.log(text.matchAll(/computer/ig), [...text.matchAll(/computer/ig)]);
```

## ΜΑΘΗΜΑ 9.3: Regular Expressions

### 1.2. Μέθοδοι του RegExp.prototype

- Το RegExp.prototype περιέχει τη μέθοδο:

Μέθοδος	Επεξήγηση
test(text)	Επιστρέφει true/false ανάλογα με το αν υπάρχει ταιριασμα της καν.έκφρασης στο text

- Εφόσον έχει οριστεί το το flag /g, τότε τα αντικείμενα RegExp αποκτούν μνήμη και:
  - Στο μέλος lastIndex επιστρέφεται η θέση του τελευταίου ταιριάσματος.
  - Αν επανακληθεί η test(..) η αναζήτηση θα γίνει μετά την θέση του τελευταίου ταιριάσματος.

#### Παράδειγμα 5: test

```
let text = "Computer Science is no more about computers " +  
  "than astronomy is about telescope";  
let regex = /computer/ig;  
  
while(regex.test(text)) {  
  console.log("Found at: " + regex.lastIndex);  
}
```

#### Σημείωση:

- Το lastIndex γίνεται ίσο με 0, μόνο όταν κληθεί η test() και το αποτέλεσμα είναι false [προσοχή, ακόμη κι αν η regex γίνει test με άλλη συμβολοσειρά, το lastIndex δεν μηδενίζεται, αλλά συνεχίζει από την προηγούμενη τιμή του]

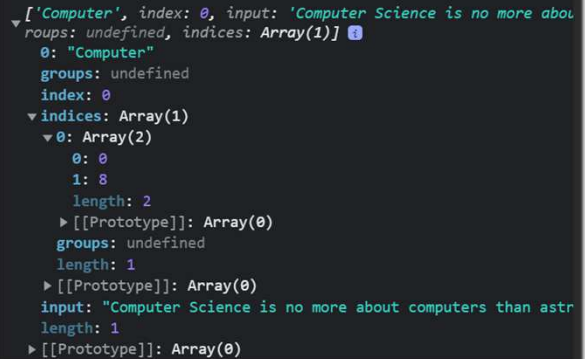
- Το RegExp.prototype περιέχει τη μέθοδο:

Μέθοδος	Επεξήγηση
exec(text)	Επιστρέφει αντικείμενο, ανάλογα με το αν υπάρχει ταιριασμα της καν.έκφρασης στο text

- Αντίστοιχη λειτουργία με την test() [απαιτείται /g και ενημερώνεται το lastIndex]
- Πρόσθετα το αντικείμενο που επιστρέφεται έχει τις ιδιότητες:
  - index: Η θέση που έγινε το ταίριασμα
  - input: Η συμβολοσειρά text
  - indices [εφόσον έχει οριστεί το /d flag] πίνακας που περιέχει τα όρια του μέρους της συμβολοσειράς που ταιριάζει με την κανονική έκφραση.

#### Παράδειγμα 6: exec

```
let text = "Computer Science is no more about computers " +  
  "than astronomy is about telescope";  
let regex = /computer/ig;  
  
while(true) {  
  let obj = regex.exec(text);  
  if (obj==null) break;  
  
  console.log(regex.lastIndex);  
  console.log(obj);  
}
```



Ταίριασμα με έναν χαρακτήρα:

re	Ταίριασμα με:
.	Ακριβώς ένας χαρακτήρας

Παράδειγμα 7: dot

```
let regexp = /a.b./;  
  
console.log("abba".search(regexp));  
console.log("baba".search(regexp));  
console.log("abbba".search(regexp));  
console.log("acbz".search(regexp));
```

Άσκηση 1:

Γράψτε κανονικές εκφράσεις για τα πρότυπα συμβολοσειρών:

- re1: Συμβολοσειρές μήκους 4 που ξεκινούν με a
- re2: Συμβολοσειρές μήκους 5 που ξεκινούν με a και τελειώνουν με bb

Έπειτα ελέγξτε αν τα παραπάνω πρότυπα περιέχονται στις συμβολοσειρές:

- aabbba
- abababab

Επανάληψη 0 ή περισσότερες φορές:

re	Ταίριασμα με:
c*	0 ή περισσότερες φορές το c (,c,cc,ccc,...)

Το c είναι είτε:

- ένας χαρακτήρας
- ακολουθία χαρακτήρων (πρέπει να εντίθενται σε παρενθέσεις), π.χ. (ab)\* ή (.a)\*

Παράδειγμα 8: star

```
console.log("Niger".search(".*er"));  
console.log("Mexico".search("Me.*"));  
console.log("Morocco".search("Mo.*ro.*cco"));  
console.log("Andorra".search(".*rr.*"));  
console.log("Burundi".search("B(u.)*ndi"));
```

Άσκηση 2:

Γράψτε κανονικές εκφράσεις για τα πρότυπα συμβολοσειρών:

- re1: Συμβολοσειρές αρτίου μήκους που ξεκινούν με a και τελειώνουν με b
- re2: Συμβολοσειρές περιττού μήκους που ξεκινούν με a και τελειώνουν με b

Έπειτα ελέγξτε αν τα παραπάνω πρότυπα περιέχονται στις συμβολοσειρές:

- aabbba, abababab

### Ειδικοί χαρακτήρες επανάληψης (quantifiers):

re	Ταίριασμα με:
c*	0 ή περισσότερες φορές το c (,c,cc,ccc,...)
c+	1 ή περισσότερες φορές το c (c,cc,ccc,...)
c?	0 ή 1 φορές
c{n,m}	n έως m φορές
c{n,}	τουλάχιστον n φορές
c{n}	ακριβώς n φορές

Το c είναι είτε:

- ένας χαρακτήρας
- ακολουθία χαρακτήρων (πρέπει να εντίθενται σε παρενθέσεις), π.χ. (ab)\* ή (.a)\*

### Παράδειγμα 9: quantifiers

```
console.log("abb".search(/ab+/));
console.log("abb".search(/ab{3,}/));
console.log("abab".search(/(ab){2}/));
console.log("abab".search(/(ab){3}/));
console.log("XabbY".search(/(a?b){2}/));
console.log("XabbY".search(/(a?b)+/));
console.log("XabbY".search(/(z?z?)*/));
```

### Οι ειδικοί χαρακτήρες επανάληψης συγκεκριμενοποιούνται με τρεις αλγοριθμικές παραλλαγές:

re	Ταίριασμα με:
c*	0 ή περισσότερες φορές το c (,c,cc,ccc,...) <b>[greedy]</b> [άπληστος, θα δοκιμάσει το ταίριασμα μεγαλύτερου μήκους, αν το επόμενο ταίριασμα της κ.ε. αποτύχει, θα δοκιμάσει το αμέσως μικρότερο ταίριασμα]
c*?	0 ή περισσότερες φορές το c (c,cc,ccc,...) <b>[reluctant]</b> [διστακτικός, θα δοκιμάσει το ταίριασμα μικρότερου μήκους, αν το επόμενο ταίριασμα της κ.ε. αποτύχει, θα δοκιμάσει το αμέσως μεγαλύτερο ταίριασμα]

- ισχύουν και για τους υπόλοιπους quantifiers (+,?,..)
- και το c πάλι είναι χαρακτήρας ή ακολουθία χαρακτήρων (πρέπει να εντίθενται σε παρενθέσεις)

### Παράδειγμα 10: quantifiers reluctant

```
console.log("aa".match(/a+/));
console.log("aa".match(/a+?/));

console.log("abaaacd".search(/ab.*cd/));
console.log("abaaacd".search(/ab.*?cd/));
```



Οι χαρακτήρες \*, +, ? κ.λπ. που είδαμε στην προηγούμενη διαφάνεια, λέγονται και **μεταχαρακτήρες (meta-characters)** γιατί αποδίδουν ένα ειδικό νόημα στην κανονική έκφραση.

- Αν θέλουμε όμως η κ.ε. μας να περιέχει κυριολεκτικά κάποιον τέτοιο χαρακτήρα, τότε θα πρέπει να προσδιορίσουμε τη χρήση του ως κανονικός χαρακτήρας.
- Αυτό καλείται **«αποφυγή χαρακτήρα» (escape character)** και γίνεται θέτοντας προηγουμένως ένα backslash (\).
  - Προσοχή στον κατασκευαστή RegExp() που παίρνει ως όρισμα συμβολοσειρά: Ο χαρακτήρας \ χρησιμοποιείται για διαφυγή σε συμβολοσειρές,, άρα **πρακτικά πρέπει να βάλουμε δύο backslash (\\)**

### Παράδειγμα 11: escape

```
console.log("a+a+".match(/a+a+/));
console.log("a+ba".match(/a\+.a+/));

let regexp = new RegExp("a\\+.a+")
console.log("a+ba".match(regexp));
```

Ενώ υπάρχουν και οι ακόλουθοι ειδικοί χαρακτήρες, για ειδική χρήση:

re	Ταίριασμα με:
\n	Αλλαγή γραμμής
\t	tab
\	To backslash

- (απαιτούν πάλι αποφυγή με ένα ακόμη backslash μπροστά, όταν χρησιμοποιούμε τον κατασκευαστή RegExp)
- και υπάρχουν και άλλοι πιο προχωρημένοι (κωδικοποίηση χαρακτήρων ως 8δικοί, 16δικοί, Unicode και ειδικοί χαρακτήρες όπως το \f (form-feed))

### Παράδειγμα 12: special characters

```
console.log("\\".match(/\\/));
console.log("\\".match(new RegExp("\\\\")));

let multiLine =
`Line 1
Line 2
Line 3`
console.log(multiLine.search(/\n/));
```

### Διάζευξη κανονικών εκφράσεων:

- Μπορούμε να επιβάλλουμε διαφορετικές περιπτώσεις στις κανονικές εκφράσεις με χρήση του συντακτικού:

re	Ταίριασμα με:
re1   re2	re1 ή re2

### Παράδειγμα 13: or

```
let text = "Computer Science is no more about computers " +
    "than astronomy is about telescope";
let regexp = /Sci.*? | comp.*? /g;
console.log([...text.matchAll(regexp)]);
```

- Δίνεται η δυνατότητα να πάρουμε τμήματα του ταιριάσματος μέσω των group

- Χρησιμοποιούμε την matchAll() με το global flag επί της κ.ε.
- Ορίζουμε ένα group ενθέτοντας το τμήμα της κανονικής έκφρασης σε παρενθέσεις
- Η αρίθμηση των groups γίνεται από αριστερά προς τα δεξιά στην κανονική έκφραση.
- Η matchAll() επιστρέφει πίνακα με τα ταιριάσματα.
  - Κάθε ταίριασμα έχει πίνακα με τα groups
  - καθώς και τα μέλη:
    - index: θέση που βρέθηκε το group
    - length: πλήθος groups

### Παράδειγμα 14: groups

```
let text = "210-11-21-222, 210-12-34-222";
let regexp = /(...)-(...)-(...)-(...)/ig;

let matches = text.matchAll(regexp);

for (let match of matches) {
    console.log(match, match.index, match.length)
    for (let group of match) {
        console.log(group);
    }
}
```

```
(5) ['210-11-21-222', '210', '11', '21', '222']
  ups: undefined
  0: "210-11-21-222"
  1: "210"
  2: "11"
  3: "21"
  4: "222"
  groups: undefined
  index: 0
  input: "210-11-21-222, 210-12-34-222"
  length: 5
  ▶ [[Prototype]]: Array(0)
```

0 5

210-11-21-222

210

11

21

222

```
(5) ['210-12-34-222', '210', '12', '34', '222']
  ups: undefined
  0: "210-12-34-222"
  1: "210"
  2: "12"
  3: "34"
  4: "222"
  groups: undefined
  index: 15
  input: "210-11-21-222, 210-12-34-222"
  length: 5
  ▶ [[Prototype]]: Array(0)
```

15 5

210-12-34-222

210

12

34



Τα ακόλουθα τμήματα κανονικής έκφρασης ορίζουν **ομάδες χαρακτήρων (character classes)**:

- Θα ταιριάζει με ένα χαρακτήρα που ανήκει στην συγκεκριμένη ομάδα

re	Ταίριασμα με:
[abc]	a ή b ή c
[^abc]	Οποιοσδήποτε χαρακτήρας εκτός των a,b,c
[a-c]	Εύρος χαρακτήρων. Ισοδύναμο του [abc]
[^a-c]	Οποιοσδήποτε χαρ/ρας εκτός των a,b,c
[a-zA-Z]	Πολλά εύρη χαρ/ρων. Ισοδύναμα [a-z] ή [A-Z]

### Παράδειγμα 15: character classes

```
console.log("psounis21@gmail.com".
  search(/[a-zA-Z1-9]{8,12}@gmail\.com/));
console.log("1.psounis21@gmail.com".
  search(/[a-zA-Z1-9]{8,12}@gmail\.com/));
console.log("psounis-21@gmail.com".
  search(/[a-zA-Z1-9_-]{8,12}@gmail\.com/));
console.log("psounis-21@gmail.com".
  search(/[a-zA-Z]+[1-9_-]*@gmail\.com/));
```

και υπάρχουν και ομάδες που έχουν οριστεί ώστε να «γκρουπάρουν» ομοειδείς χαρακτήρες:

re	Ταίριασμα με:
\d	ψηφίο [0-9]
\D	όχι ψηφίο [^0-9]
\s	whitespace [ \t\n\r\x0B\f]
\S	όχι whitespace [^\s]
\w	word character [a-zA-Z_0-9]
\W	όχι word character

### Παράδειγμα 16: predefined character classes

```
let pattern = /([0-1]\d|2[0-3]):[0-5]\d/;
console.log("01:49".search(pattern));
console.log("11:69".search(pattern));
console.log("24:14".search(pattern));
console.log("23:59".search(pattern));

pattern = /([0-1]\d|2[0-3]):[0-5]\d(\.\d{1,4})?/;
console.log("01:49".search(pattern));
console.log("11:19.123".search(pattern));
console.log("23:14.12345".search(pattern));
console.log("25:59".search(pattern));
```

## ΜΑΘΗΜΑ 9.3: Regular Expressions

### 2.6. Ταίριασμα στα άκρα

Μέσω ειδικών χαρακτήρων μπορούμε να κάνουμε ταίριασμα σε άκρα της συμβολοσειράς.

- Οι παρακάτω επιβάλλουν το άκρο της συμβολοσειράς να έχει συγκεκριμένη μορφή:

re	Ταίριασμα με:
^	αρχή
\$	τέλος

#### Παράδειγμα 17: boundary

```
let text = "XXwordXX";
let regexp = /word/
let regexp2 = /^word/
let regexp3 = /word$/
let regexp4 = /^word$/

console.log(text.search(regexp));
console.log(text.search(regexp2));
console.log(text.search(regexp3));
console.log(text.search(regexp4));

let pattern = /([0-1]\d | 2[0-3]):[0-5]\d(\.\d{1,4})?/;
console.log("23:14.12345".search(pattern));
pattern = /^([0-1]\d | 2[0-3]):[0-5]\d(\.\d{1,4})?$/;
console.log("23:14.12345".search(pattern));
```

- Ενδιαφέρον έχει και οι ακόλουθοι δύο ειδικοί χαρακτήρες

re	Ταίριασμα με:
\b	άκρο λέξης. Επιβάλλεται στην αρχή, στο τέλος ή και στα δύο.
\B	όχι άκρο λέξης. Επιβάλλεται στην αρχή, στο τέλος ή και στα δύο.

#### Παράδειγμα 18: boundary\_word

```
let text = "Computer Science is no more about computers " +
"than astronomy is about telescope";

let regexp = /\bscope/i;
console.log(text.search(regexp));

let regexp2 = /\Bscope/i;
console.log(text.search(regexp2));

let regexp3 = /(\bcomputer\b)/ig;
console.log([...text.matchAll(regexp3)]);
```