Llenguatges de Marques i Sistemes de Gestió d'Informació

5.2. Annex Expressions regulars

IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2024-25

Índex

1 Expressions regulars

Les **expressions regulars**, comunment conegudes com ***"regex"*** o ***"RegExp"***, són cadenes de text amb un format especial que s'utilitzen per trobar **patrons** al text. Les expressions regulars són una de les eines més potents disponibles per al processament i manipulació de text. Per exemple, es pot utilitzar per verificar si el format de les dades (nom, correu electrònic, número de telèfon, etc.) introduïdes per l'usuari, és correcte o no, trobar o substituir una cadena dins del contingut del text, etc.

1.1 Funcions de JavaScript per a regExs

En JavaScript tenim vàries funcions per a treballar amb expressions regulars.

La que utilitzarem majoritàriament per a validar cadenes és **`regex.test(str)`**:

```
if ( /^\d*$/.test(valor) ) {
  console.log('El valor llegit és un número');
} else {
  console.log('El valor llegit NO és un número');
}
```

El mètode **`str.match(regex)`** cerca totes les aparicions de l'expressió regular en la cadena `str`. Si l'expressió té el flag **g** (`global`, per a buscar totes les coincidències), retorna un array amb totes les coincidències:

```
let regexp = /ola/g;
let str = "Hola caracola";
let result = str.match(regexp);
console.log(result); // Array [ "ola", "ola" ]
console.log(result.length); // 2

let str2 = "Adeu";
let result2 = str2.match(regexp);
console.log(result2); // null
```

Si l'expressió no du el flag **g**, `match` retorna un array amb la cadena trobada i la posició de la primera coincidència.

Finalment, **`str.replace(regexp, repl)`** reemplaça totes les coincidències de l'expressió `regexp` en la cadena `str` amb `repl`. Si està el flag g, reemplaçarà totes les coincidències, si no, només la primera:

```
let regexp = /ola/g;
let str = "Hola caracola";
```

```
let result = str.replace(regexp, "olo");
console.log(result); // "Holo caracolo"
```

> **La mateixa regexp utilitzada globalment amb `regexp.test()` pot fallar**, ja que JavaScript utilitza la propietat `lastIndex`, que guarda la última posició de cerca. Per a evitar-ho es pot reiniciar `lastIndex` (`regexp.lastIndex = 0`) o usar `str.match()` en lloc de `regexp.test()` per a comprovar que el resultat no és `null`:

```
if ( valor.match(/^\d*$/) ) {
    console.log('El valor llegit és un número');
} else {
   console.log('El valor llegit NO és un número');
}
```

1.2 Expressions simples

Les expressions regulars en JavaScript es representen entre 2 barres **`/`**. Les barres funcionen de manera similar a les cometes per als strings. A més, després de la segona barra podem incloure **`flags`** o banderes que modifiquen la cerca de l'expressió en la cadena de text.

Exemples d'expressions:

```
let regexp = /hola/; // Sense flags
let regexp = /hola/gm; // Amb flags
```

També podem usar una sintàxi més llarga:

```
let regexp = new RegExp('hola', 'gm'); // equival a /hola/gm
```

Els caràcters que escrivim entre les 2 barres és la cadena de text que buscarà. Per exemple:

```
let regexp = /ola/;
regexp.test("Hola caracola"); // true
```

Retorna `true` perquè ha trobat l'expressió `ola` a partir del segon caràcter.

Amb el flag `i`, la cerca es fa ignorant majúscules i minúscules:

```
let regexp = /hola/;
regexp.test("Hola caracola"); // false

regexp = /hola/i;
regexp.test("Hola caracola"); // true
```

1.3 Classes de caràcters

Algunes classes de caràcters com ara dígits, lletres i espais en blanc s'utilitzen amb tanta freqüència que tenen abreviatures. La taula següent enumera algunes abreviatures predefinides:

Abreviatura	Significat
`.`	Qualsevol caràcter excepte nova línia (\n)
`\d`	Qualsevol dígit
`\D`	Qualsevol caràcter NO dígit
`\w`	Qualsevol caràcter alfanumèric
`\W`	Qualsevol caràcter NO alfanumèric
`\s`	Qualsevol espai, tabulador, nova línia (\n)
`\S`	Qualsevol caràcter NO espai, tabulador, nova línia (\n)

El següent exemple reemplaça tots els espais d'una cadena per guions:

```
let result = "pim pam pum".replace(/\s/g, '-');
console.log(result); // pim-pam-pum
```

Els claudàtors `[ ]`, també representen classes definides per nosaltres. Significa que es buscarà qualsevol dels caràcters entre claudàtors:

```
let regexp = /[ola]/g;
let result = "Hola caracola".match(regexp);
console.log(result); // Array(8) [ "o", "l", "a", "a", "a", "o", "l", "a" ]
```

Dins dels claudàtors es pot usar el signe de negació ^ per a que busque tots els caràcters excepte els que estàn entre claudàtors. I també es pot utilitzar el guió – per a que busque tots els caràcters entre els 2 indicats. Alguns exemples:

RegExp	Significat
[abc]	Coincideix amb a, b, o c
[^abc]	Coincideix amb qualsevol caràcter excepte a, b, o c
[a-z]	Qualsevol lletra en minúscula, de la a a la z
[A-Z]	Qualsevol lletra en majúscula, de la A a la Z

RegExp	Significat
[a-zA-Z]	Qualsevol lletra, en majúscula o minúscula
[0-9]	Qualsevol dígit, equival a \d

1.4 Quantificadors de repeticions

Si volem fer coincidir més d'un caràcter, per exemple si volem trobar paraules que contenen una o més instàncies de la lletra p, o paraules que contenen almenys dues p, hem d'utilitzar els **quantificadors**. Amb els quantificadors podem especificar quantes vegades ha de coincidir un caràcter d'una expressió regular.

Alguns exemples:

RegExp	Significat
`a+`	Una o més ocurrències de la lletra a (a, aa, aaa ...)
`a*`	Zero o més ocurrències de la lletra a (coincidiria amb tots els caràcters!)
`la*`	Zero o més ocurrències de la cadena `la`
`a?`	Zero o una ocurrències de la lletra a (coincidiria amb tots els caràcters!)
`a{2}`	Exactament 2 ocurrències de la lletra a (`aa`)
`a{2,}`	2 o més ocurrències de la lletra a (`aa`, `aaa` ...)
`\w{4}`	Qualsevol cadena de 5 caràcters alfanumèrics
`\d{9}`	Qualsevol cadena de 9 dígits

Exemple:

```
let result = "Mamaaaaa".replace(/a{2,}/g, 'i');
console.log(result); // Mami
```

1.5 Altres modificadors

És comú usar la barra **|** junt amb parèntesis **( )** per a cercar una cadena o altra de les indicades entre els parèntesis:

```
let regexp = /(hola|cola)/gi;
let result = "Hola caracola".match(regexp);
console.log(result); // Array(2) [ "Hola", "cola" ]
```

Els parèntesis també es poden usar junt amb el quantificadors de repeticions per a buscar repeticions de cadenes. Per exemple, `va+` significa la lletra `v` seguida d'una o vàries `a`, però `(va)+` significa una o vàries rpeticions de la cadena `va`.

Un altre modificador molt útil és el **límit de paraula `\b`**. S'utilitza per a buscar paraules senceres, sense importar si estan al principi o al final de la cadena:

```
let regexp = /\bJava\b/gi;
let result = "Hola Java!".match(regexp);
console.log(result); // Array [ "Java" ]

result = "Hola JavaScript!".match(regexp);
console.log(result); // null
```

1.6 Àncores de posició

Quan volem buscar una expressió al principi o al final d'una cadena o de línies de textos llargs, podem usar les àncores. Les més comuns son `^` que busca al principi de la cadena i `$` que busca coincidències pel final. Per exemple:

```
let regexp = /^hola/gi;
regexp.test("Hola caracola"); // true

regexp = /hola$/i;
regexp.test("Hola caracola"); // false

regexp = /ola$/i;
let result = "Hola caracola".match(regexp);
console.log(result); // Array [ "ola" ]
```

Ací podem usar el flag **`m`** (multi-línea) per a tractar cada línia d'un text com a una cadena diferent, sent útil per a tractar llistats que terminen en un salt de línia.

1.7 Caràcters d'escape

Quan volem utilitzar en l'expressió regular els caràcters especials com `/`, `( )`, `[ ]`, etc. hem d'*escapar-los* per a que siguen interpretats com a caràcters normals. La forma d'escapar-los és per mitjà d'una barra invertida `\`:

`\[, \], \\, \/, \^, \$, \., \|, \?, \*, \+, \(, \),`

Si estem utilitzant `new RegExp` per a crear les expressions, no hem d'escapar el caràcter `/`, però en canvi hem d'escapar tots els caràcters `\`, ja que les cadenes utilitzen aquest caràcter com a caràcter d'escape.

1.8 Resum de flags

Ja hem vist que els flags més utilitzats són `g`, `m` i `i`, però JavaScript té algun flag més. La següent taula resumeix tots els flags de JavaScript:

Flag	Significat
`i`	La cerca no distingeix entre majúscules i minúscules
`g`	La cerca troba totes les coincidències, sense `g`, només es torna la primera coincidència.
`m`	Mode multi-línea
`d`	Habilita el mode *dotall*, que permet que un punt `.` coincidisca amb el caràcter de nova línia `\n`
`u`	Permet el suport complet d'Unicode
`y`	Mode adhesiu, cerca en una posició exacta indicada per `lastIndex`