# Espressioni regolari in Javascript (RegExp)

**Antonio Lioy**
**< lioy @ polito.it >**
***Politecnico di Torino***
***Dip. Automatica e Informatica***

***(estensione di materiale preparato da Andrea S. Atzeni)***

## Le espressioni regolari

- **definiscono un lessico = insieme di "parole" lecite**
- **… specificando**
  - i caratteri
    - alfabetici (A,a)
    - numerici (1,2, ...)
    - segni di interpunzione (. : , ; ...)
  - le sequenze ammissibili
    - es. "una sequenza di uno o più caratteri A"
- **… con un formalismo "compatto" (metacaratteri)**
  - es. $A^+$ => le sequenze A, AA, AAA, AAAA, ...

## Un po' di storia

- **termine coniato dal matematico S. Kleene ('50)**
- **sviluppato all'interno della teoria dei linguaggi**
  - stessa potenza espressiva di macchine a stati deterministiche
- **molto usate in ambiente Unix**
- **utilizzi tipici:**
  - formulazione di potenti ricerche testuali (es. grep)
  - automazione di editing testuale (es. sed)

## Insiemi di caratteri

- **[…] per includere uno qualsiasi dei caratteri in parentesi**
  - possibile specificare singoli caratteri o intervalli di caratteri adiacenti (es. A-Z per indicare tutte le lettere alfabetiche maiuscole)
  - es. [a-zABC] riconosce una qualsiasi lettera alfabetica minuscola oppure A, B, o C
- **[^…] per escludere uno qualsiasi dei caratteri in parentesi**
  - es. [^0-9] riconosce qualsiasi carattere non numerico

## Metacaratteri

- **\ (cosiddetto “escape”) per**
  - segnalare sequenze speciali
  - considerare caratteri speciali come caratteri normali
  - es. \[ cerca il carattere [
- **simboli speciali per identificare un carattere …**
  - \d = numerico, ossia [0-9]
  - \D = non numerico, ossia [^0-9]
  - \s = equivalente a "spazio" (blank CR LF FF HT VT)
  - \S = non equivalente a "spazio"
  - \w = alfanumerico o _, ossia [a-zA-Z0-9_]
  - \W = non alfanumerico o _, ossia [^a-zA-Z0-9_]

## Metacaratteri (cont)

- **| (or logico) per esprimere un’alternativa tra due espressioni**
  - sintassi: regexp1 | regexp2
  - es. "A|B" riconosce sia il carattere A sia il carattere B
- **. indica un carattere qualsiasi**
  - es. "R.E" riconosce una stringa di tre caratteri che inizi con R e termini con E, quali "RaE", "RAE" o "RiE"

## Inizio e fine stringa

- **^ corrisponde all'inizio della stringa**
  - sintassi: ^regexp
  - es. "^buongiorno" riconosce “buongiorno, eccomi qui!” ma non “Dimmi almeno buongiorno”
- **$ corrisponde alla fine della stringa**
  - sintassi: regexp$
  - es. "200$" riconosce "nell'anno 1200" ma non "nell'anno 2000"
- **esempio (copia un file eliminando le righe vuote):**

```
grep -v "^$" file1 > file2
```

## Raggruppamenti

- **( … ) per raggruppare espressioni e creare clausole complesse**
  - sintassi: (regexp1<c>regexp2)
  - es. "pa(ss|zz)o" riconosce "passo" oppure "pazzo"
- **{N}**
  **{Nmin,}**
  **{Nmin,Nmax}**
  **per specificare la numerosità (esatta, minima, minima e massima)**
  - sintassi: regexp{…}
  - es. "\d{2,4}" riconosce numeri composti da almeno due cifre ed al massimo da quattro, quali “23”, “655” o “4345” ma non “2” o “34567” o “2+3”

## Numero di occorrenze

- *** indica zero o più occorrenze di un’espressione**
  - sintassi: regexp*
  - es. (ab)* riconosce sequenze di "ab" di qualsiasi lunghezza, come "ab" "abab" o anche ""
- **+ indica una o più occorrenze di un’espressione**
  - sintassi: regexp+
  - es. (ab)+ riconosce sequenze di "ab" di qualsiasi lunghezza, come "ab" "abab" ma non ""
- **? indica zero o al più una occorrenza**
  - sintassi: regexp?
  - es. "Mar(i)?a" riconosce le sequenze “Mara”, “Maria”, e “Mariangela” ma non "Mariiiiia!"

## Le espressioni regolari in Javascript

- **possono essere create**
  - come sequenze letterali
    - es. re = /ab+c/
    - valutata quando viene caricato lo script
    - più veloce se l'espressione rimane costante
  - come costruttore dell'oggetto RegExp
    - es. re = new RegExp("ab+c")
    - valutata a runtime
- **attenzione al carattere backslash:**

```
re = /\d{2}/
re = new RegExp("\\d{2}")
```

## Flag delle regexp in JS

- **possibile indicare dei flag quando si crea la regexp**
  - re = / pattern / flags
  - re = new RegExp ("pattern", "flags")
- **possibili flag:**
  - i = case-insensitive (ignora la differenza tra caratteri maiuscoli e minuscoli)
  - g = global (non si ferma al prima match ma prosegue la ricerca su tutta la stringa)
  - m = multiline (se la stringa contiene terminatori di linea allora ogni riga è considerata una stringa diversa ai fini dei caratteri ^ e $)
- **esempio: re = /ciao/i**

## Metodi JS per regexp (base)

- **string.search( regexp )**
  - restituisce l'indice iniziale della (prima) stringa trovata, oppure -1
- **regexp.test( stringa )**
  - restituisce true se all'interno della stringa compare l'espressione, false altrimenti
- **esempio:**

```
var s = "ciao,mamma"
var re = /\W/
n = s.search(re) // restituisce 4
x = re.test(s)   // restituisce true
```

## Metodi JS per regexp (avanzati)

- **le espressioni regolari possono essere usate per identificare una o più sottostringhe (e cambiarle o salvarle in un array)**
- **string.match( regexp )**
  - restituisce le stringhe trovate oppure null
- **string.replace( regexp, new)**
  - restituisce la stringa di partenza con new al posto delle parti selezionate dall'espressione regolare
- **string.split( regexp )**
  - restituisce le stringhe identificate in base alla regexp
- **regexp.exec( string )**
  - restituisce le stringhe identificate in base alla regexp

## Validazione dei dati di un form

```
function verifica(d) {
  var expr=/^\d{2}-\d{2}-\d{4}$/
  if (expr.test(d))
    window.alert(d+": formato corretto")
  else
    window.alert(d+": formato errato")
}
```

```
<form>
  data (gg-mm-aaaa) <input type="text" name="data">
  <input type="button" value="verifica"
    onclick="verifica(data.value)">
</form>
```

checkdata.html

## Esempio (password)

- **possibile formato di una password:**
  - alfanumerica, minimo 8, massimo 16 caratteri
  - almeno una maiuscola, una minuscola ed una cifra
- **funzione che restituisca true se la pwd è corretta, false altrimenti**

```
function is_pwd_OK(p) {
  return (
    /^[a-zA-Z0-9]{8,16}$/.test(p) &&
    /[A-Z]/.test(p) &&
    /[a-z]/.test(p) &&
    /[0-9]/.test(p) )
}
```

## Esempio (numero naturale)

- **numero naturale, scritto eventualmente col punto come separatore delle migliaia**
- **funzione che restituisca -1 se il formato è errato, altrimenti il numero di partenza senza gli eventuali punti (ad esempio 21.876 deve diventare 21876)**

```
function n_naturale(x) {
  re = /^\d{1,3}(\.?\d{3})*$/
  if (re.test(x))
    return x.replace(/\./g,"")
  else
    return -1
}
```

## Test dei valori

- **il test dei programmi è un'arte...**
- **...ed anche una scienza**
- **testare i casi "positivi"...**
  - tutti se possibile (raramente)
  - sottoinsiemi sensati (max, min, ...)
- **...ma anche quelli "negativi"**
  - tipi diversi, misti, valori atipici, ...