UNIVERSITÀ DEGLI STUDI DI PALERMO

dipartimento di ingegneria unipa

Gestione degli errori, date, espressioni regolari, JSON

Corso di Programmazione Web e Mobile

a.a. 2021/2022

Prof. Roberto Pirrone

LABORATORIO DI INTERAZIONE UOMO-MACCHINA
CHILAB

Sommario

- Gestione degli errori
  - `'use strict'`
  - `try … catch`
  - Errori definiti dall'utente
- Classe Date e operazioni sulle date
  - Gestione degli intervalli di tempo
- Classe RegExp
- JSON

Gestione degli errori

- La direttiva `'use strict'` rinforza il controllo del contesto di esecuzione forzando tutta l'elaborazione a seguire *il contesto più interno* di riferimento

  - Non è possibile dichiarare variabili locali senza `let` perché viene impedito che queste vengano automaticamente apparentate al contesto globale, come accade nella modalità «non ristretta»

  - L'esecuzione di funzioni o metodi senza contesto (cioè senza esplicito oggetto chiamante al di fuori del contesto generale) forza `this` ad assumere il valore `undefined` e non il valore del contesto globale

Gestione degli errori

```
ciao!
ciao!
ciao!
ciao!
ciao!
Fred Flinstone

 for (i = 0; i < 5; i++)
             ^


ReferenceError: i is not defined

 this.name = name;
               ^


TypeError: Cannot set property
'name' of undefined
```

senza ‘use strict’

```
'use strict';

function pippo() {
        for (i = 0; i < 5; i++)
        console.log('ciao!');
}

function People(name, surname) {
        this.name = name;
        this.surname = surname;
}

pippo();
let fred = People('Fred', 'Flinstone’);
console.log(`${fred.name} ${fred. surname}`);
```

Gestione degli errori

- `throw` lancia l'errore con il messaggio dedicato
  - Lo si trova nella stack trace quando l'errore si verifica
  - L'esecuzione si interrompe
- I blocchi `catch` e `finally` sono opzionali

```
try {
        code
        throw new Error(message);
}
catch(error){
        recovery action
}
finally{
        code to be executed anyway
}
```

Gestione degli errori

- L'utilizzo di `instanceof` su un tipo di errore definito dall'utente consente il *catch selettivo*

```
class MyError extends Error {}

try {
        code
        throw new MyError(message);
}
catch(error){
        if(error instanceof MyError)
                recovery action
        else
                throw error;
}
finally{
        code to be executed anyway
}
```

Date

- La classe `Date` gestisce la data, l'ora e il fuso orario corrente nonché consente di generare date definite dall'utente

- E' possibile quindi implementare calcoli di intervalli di tempo

- La rappresentazione interna della data è il numero di millisecondi trascorsi dal 1 gennaio 1970.

Date

```
> let now = new Date()
undefined
> now.
now.__defineGetter__        now.__defineSetter__        now.__lookupGetter__
now.__lookupSetter__        now.__proto__               now.hasOwnProperty
now.isPrototypeOf           now.propertyIsEnumerable

now.constructor             now.getDate                 now.getDay
now.getFullYear             now.getHours                now.getMilliseconds
now.getMinutes              now.getMonth                now.getSeconds
now.getTime                 now.getTimezoneOffset       now.getUTCDate
now.getUTCDay               now.getUTCFullYear          now.getUTCHours
now.getUTCMilliseconds      now.getUTCMinutes           now.getUTCMonth
now.getUTCSeconds           now.getYear                 now.setDate
now.setFullYear             now.setHours                now.setMilliseconds
now.setMinutes              now.setMonth                now.setSeconds
now.setTime                 now.setUTCDate              now.setUTCFullYear
now.setUTCHours             now.setUTCMilliseconds      now.setUTCMinutes
now.setUTCMonth             now.setUTCSeconds           now.setYear
now.toDateString            now.toGMTString             now.toISOString
now.toJSON                  now.toLocaleDateString      now.toLocaleString
now.toLocaleTimeString      now.toString                now.toTimeString
now.toUTCString             now.valueOf
```

Date

- I giorni della settimana e i mesi dell'anno sono indicizzati numericamente e quindi è sempre necessario implementare delle tabelle di lookup per associare l'indicizzazione alle stringhe con i nomi dei mesi o dei giorni.

- Le differenze tra date possono essere ottenute tramite la differenza tra i valori in millisecondi della rispettiva rappresentazione interna con il metodo `getTime()`

    - È necessario scomporre esplicitamente il valore ottenuto negli anni, mesi e giorni tramite il calcolo dei millisecondi che compongono un anno/mese/giorno

Intervalli di tempo

- In Javascript è possibile gestire la chiamata temporizzata di una funzione
  - Timeout → chiamata di una funzione dopo un certo intervallo di tempo
  - Interval → chiamata di una funzione ad ogni intervallo di tempo

```
let timeoutID = setTimeout(funzione, millisecondi, [argomenti])
clearTimeout(timeoutID)
```

Intervalli di tempo

- In Javascript è possibile gestire la chiamata temporizzata di una funzione
  - Timeout → chiamata di una funzione dopo un certo intervallo di tempo
  - Interval → chiamata di una funzione ad ogni intervallo di tempo

```
let intervalID = setInterval(funzione, millisecondi, [argomenti])
clearInterval(intervalID)
```

Espressioni regolari

- Javascript gestisce le espressioni regolari attraverso la classe `RegExp` che fornisce una interfaccia per il parsing delle stringhe rispetto all'espressione regolare stessa e restituisce tutti i match

- Il letterale delle espressioni regolari è definito nel seguente modo

```
let  myRegExp = /corpo dell'espressione/modificatori
```

`g` → match globale ; `i` → case insensitive; `m` → match multilinea

Espressioni regolari

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

Credits W3Schools.com

Espressioni regolari

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b
\B	Find a match, but not at the beginning/end of a word
\0	Find a NULL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\udddd	Find the Unicode character specified by a hexadecimal number dddd

Credits W3Schools.com

Espressioni regolari

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X* *n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*

Credits W3Schools.com

Espressioni regolari

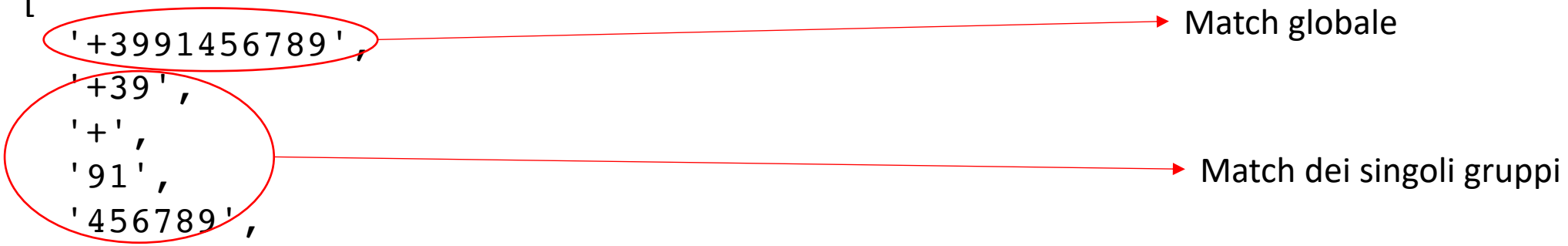
```
> let tel = /^((\+|00)[1-9]{2}|0)?([1-9]{2,3})([0-9]{6,10})$/g
undefined
> tel.
tel.__defineGetter__      tel.__defineSetter__      tel.__lookupGetter__
tel.__lookupSetter__      tel.__proto__             tel.hasOwnProperty
tel.isPrototypeOf         tel.propertyIsEnumerable  tel.toLocaleString
tel.valueOf

tel.compile               tel.constructor           tel.dotAll
tel.exec                  tel.flags                 tel.global
tel.ignoreCase            tel.multiline             tel.source
tel.sticky                tel.test                  tel.toString
tel.unicode

tel.lastIndex
```

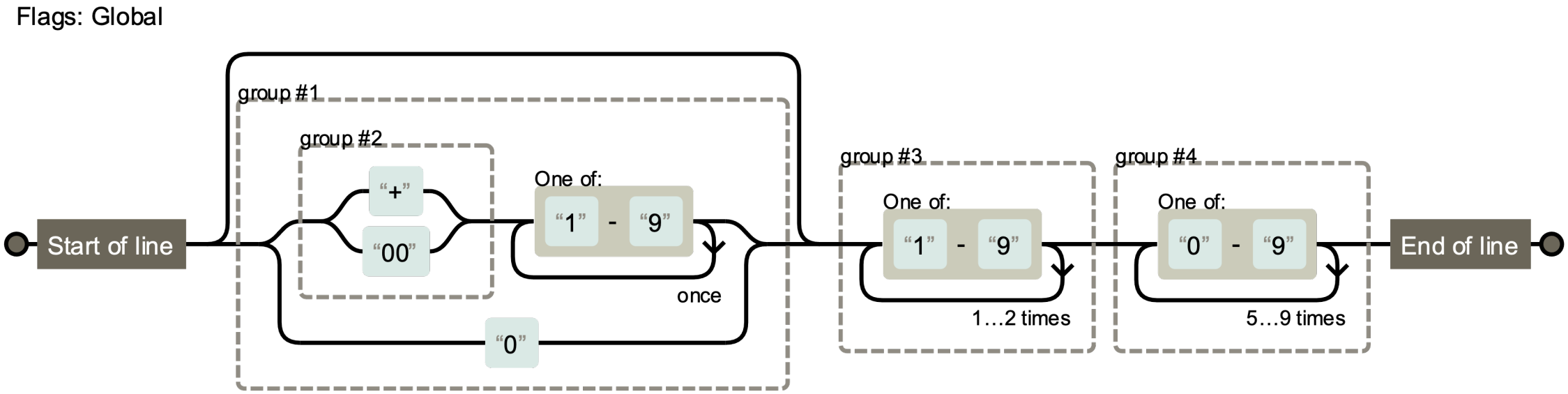
Espressioni regolari

```
> tel.test('+3991456789')
true
> tel.exec('+3991456789')
[
  '+3991456789',
  '+39',
  '+',
  '91',
  '456789',
  index: 0,
  input: '+3991456789',
  groups: undefined
]
> '+3991456789'.search(tel)
0
```



Espressioni regolari

```
/^((\+|00)[1-9]{2}|0)?([1-9]{2,3})([0-9]{6,10})$/g
```



JSON

- JSON significa *Javascript Object Notation*
- È il formato testuale di interscambio di oggetti Javascript sulla rete
- È un formato strutturato ed è un ottimo sostituto di XML
- La classe `JSON` offre un'interfaccia fatta da due metodi:

  `parse()` → parsing di una stringa JSON in oggetto Javascript

  `stringify()` → serializzazione di un oggetto Javascript come stringa

JSON

- JSON ammette i seguenti elementi al suo interno:
  - Stringhe
  - Numeri
  - Oggetti composti da elementi validi in JSON
  - Array
  - Booleani
  - `null`

JSON

```
> let myObj = {name:'Roberto',
... surname:'Pirrone',
... age:54,
... isAProfessor: true,
... appointments: {headOfLab: true, headOfDept: false, PhDTutor: true},
... classes: ['ProWebMo', 'BigData']}
Undefined
> let jsonObj=JSON.stringify(myObj)
undefined
> jsonObj
'{"name":"Roberto","surname":"Pirrone","age":54,"isAProfessor":true,
"appointments":{"headOfLab":true,"headOfDept":false,"PhDTutor":true},
"classes":["ProWebMo","BigData"]}'
> JSON.parse(jsonObj)
{
  name: 'Roberto',
  surname: 'Pirrone',
  age: 54,
  isAProfessor: true,
  appointments: { headOfLab: true, headOfDept: false, PhDTutor: true },
  classes: [ 'ProWebMo', 'BigData' ]
}
```