JavaScript

- Altre funzionalità
  - Oggetti e funzioni
  - Eccezioni
  - Regular Expression
  - Supporto Object Oriented ES6
- Progetto di riferimento
  - https://github.com/egalli64/nesp *(modulo 3c)*
    - Node.js
    - VS Code

Oggetto

- Struttura definita dal *prototype* del suo costruttore
  - Proprietà nell'oggetto creato
  - Ereditarietà con Object alla base della gerarchia
  - "**this**" è il reference all'oggetto che viene creato usando "**new**"
- La relazione può essere indicata nel costruttore del child via call()
  - Primo parametro: this
  - Gli altri sono quelli del costruttore
- In alternativa si può chiamare apply()
  - Primo parametro: this
  - Secondo parametro: array dei parametri del ctor

```
function Person(first, last) {
    /* … */
}

function PersonEx(first, middle, last)
{
    Person.call(this, first, last);
    /* … */
}
```

Funzioni

- L’oggetto arguments
  - accesso ai parametri per indice (ma non è un vero array)
  - In ES6 si preferisce usare ‘rest’
- Funzione anonima “freccia” ES 6
  - Sintassi compatta
  - Non ha un proprio “this”, non può essere usata come costruttore
- Default per parametri ES 6
  - X = 0
- Parametro ‘rest’ ES 6
  - … va
  - È un array standard

```
let f2 = () => 'hello from f2';

let f3 = () => {
    console.log('hello from f3');
    return 'done';
}
```

Regular Expression

- Sequenza di caratteri che identifica un pattern, ad esempio: un CAP è una stringa di cinque cifre
  - Creazione di un oggetto apposito → new RegExp('world', i); oppure: stringa delimitata da slash più opzioni → **/**world**/** /world/i
- Indice della prima occorrenza del pattern nella stringa: str.search(regex)
- Check booleano per un match nella stringa regex.test(str)
- Uso di parentesi quadre per match di un carattere con più alternative
  - **[**aeiou**]** → una vocale, [a-z] → un carattere alfabetico minuscolo
    - Abbreviazioni: **\d** → [0-9], **\w** → [A-Za-z0-9_], **\s** → spazi, tab, newline, …
  - L'accento circonflesso '**^**' in questo contesto nega la condizione: ^\d, [^0-9] → non cifra
    - Abbreviazioni: **\D** → ^\d, **\W** → ^\w, **\S** → ^\s
- Quantificatori:
  - Parentesi graffe **{**n, m**}** tra n e m (positivi e ordinati) ripetizioni – varianti {n}, {n, }
  - Punto di domanda '**?**' per 0 o 1, asterisco '*****' per 0 o più, più '**+**' per almeno uno, equivalente a {1,}
  - Versioni non-greedy di * e +, con un punto di domanda postfisso
- Il punto '**.**' indica un carattere qualunque
- Ancore all'inizio e fine della stringa, accento circonflesso '**^**' e dollaro '**$**'

Eccezioni

- Gestioni rigorosa degli errori
- Se l’eccezione non viene gestita, lo script termina

```
function indexToMonthName(index) {
    if (!Number.isInteger(index) || index < 1 || index > 12) {
        throw 'invalid month number: ' + index;
    }

    let months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
        'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
    return months[index - 1];
}
```

```
try {
    console.log(indexToMonthName(1));
    console.log(indexToMonthName(12));
    console.log(indexToMonthName(0));
} catch (exc) {
    console.log(exc);
} finally {
    console.log('done');
}
```

Destrutturazione

Estrazione di informazioni da array/oggetti in variabili distinte (ES 6)

```
let data = [1, 2, 3, 4, 5];
let [first, second] = data; // i primi due elementi dell'array
let [a, , c, ...va] = data; // primo, terzo, e tutti gli altri
```

sintassi "spread"

```
let x = 12;
let y = 24;

[x, y] = [y, x]; // swap
```

```
let obj = { a: 42, b: true };
let { a, b } = obj;
```

```
let obj = { a: 42, b: true };
let { a: age, b: flag } = obj; // estrazione con nuovi nomi
```

Array – altri metodi

- Array con dimensione: Array(size)
- Inizializzazione: fill() (ES 6)
- Nuovo array
  - da un array/iterable: Array.from() (ES 6)
  - specificando un intervallo [begin, end): slice()
- Ordine in-place: sort()
- Inversione in-place: reverse()
- Applicazione di una funzione su tutti gli elementi
  - Esecuzione semplice: forEach()
  - Tutti gli elementi soddisfano una condizione? every()
  - Nuovo array
    - filtrato: filter()
    - modificato: map()

```
let array = new Array(5); // [undefined, ...]
array.fill(0); // [0, …]

let splits = Array.from('hello');

let sliced = array.slice(1, 3);

splits.sort((l, r) => l == r ? 0 : l < r ? 1 : -1);

splits.reverse();

splits.forEach(x => console.log(x));

if(slices.every(s => s == 'l')) { /*… */ }

let filters = splits.filter(s => s > 'h');

let uppers = splits.map(s => s.toUpperCase());
```

Template literals (o strings)

- ES 6
- Stringhe che gestiscono espressioni interne e in cui possiamo andare a capo esplicitamente invece di usare ‘\n’
- Delimitate da accenti gravi (backtick alt-96 ‘`’)
- Possono contenere placeholder, nel formato ${expr}

```
let x = 12;
let y = 24;
console.log(`Sum is ${x + y}`);
```

Altri loop

for … in
(oggetti)

for … of
(iterabili)

ES 6

Array.forEach()

```
let props = { a: 1, b: 2, c: 3 };
for (let prop in props) {
    console.log(`${prop} is ${props[prop]}`);
}
```

```
let ys = [1, 2, 3, 4, 5, 6];
for (let y of ys) {
    console.log(y);
}
```

```
ys.forEach((y) => {
    console.log(y);
});
```

Set e Map

- collezioni iterabili in ordine di inserimento
- Set (ES 6)
  - valori unici (verifica via ‘===’ ma NaN considerato === NaN)
  - add(), clear(), delete(), forEach(), has(), values(), size
- Map (ES 6)
  - Relazione chiave → valore
  - Le chiavi possono essere di qualunque tipo
  - clear(), delete(), entries(), forEach(), get(), has(), keys(), set(), values()

class

ES6

Un solo ctor!

```
class Person {
    constructor(first, last) {
        this.first = first;
        this.last = last;
    }

    fullInfo() {
        return this.first + ' ' + this.last;
    }
}
```

```
let p = new Person('Tom', 'Jones');
```

Pseudoproprietà: get e set

ES6

```
class Person {
    // ...

    get fullName() {
        return this.first + ' ' + this.last;
    }

    set fullName(name) {
        let buffer = name.split(' ');
        this.first = buffer[0];
        this.last = buffer[1];
    }
}
```

```
let p = new Person('Tom', 'Jones');
p.fullName = 'Bob Hope';
console.log(p.fullName);
```

Static

ES6

```
class Person {
    // ...

    static merge(p1, p2) {
        return new Person(p1.first + p2.first, p1.last + p2.last)
    }
}
```

```
let tom = new Person('Tom', 'Jones');
let bob = new Person('Bob', 'Hope');

console.log(Person.merge(tom, bob).fullName);
```

Ereditarietà

```
class Employee extends Person {
    constructor(first, last, salary) {
        super(first, last);
        this.salary = salary;
    }

    fullInfo() {
        return super.fullInfo() + ': ' + this.salary;
    }
}
```


```
let jon = new Employee('Jon', 'Voight', 2000);
console.log(jon.fullInfo());
```