

JavaScript

- Il linguaggio
 - Nato nel browser come integrazione ad HTML
 - Ora anche uso indipendente: Node JS
- Supporto Object Oriented ES6
- Progetto di riferimento
 - <https://github.com/egalli64/nesp> (*modulo 3a*)

JavaScript

- Linguaggio di programmazione interpretato, debolmente e dinamicamente tipizzato, multi-paradigma, imperativo, object-based, funzionale, event-driven
- Nato nel 1995 (Brendan Eich @ Netscape) per aggiungere funzionalità alla coppia HTML-CSS, è ora utilizzato anche esternamente ai browser
- Dal 1997 ECMA ne coordina lo sviluppo, con il nome ufficiale di ECMAScript
 - ES 5 (2009)
 - ES 6 – nome ufficiale EcmaScript 2015 (ES2015)
 - ...
 - ES 10 (2019)
- Sostanzialmente diverso da Java

Node JS

- Piattaforma per server app in JavaScript
 - Ben supportata da VS Code
- <https://nodejs.org/en/download/> (LTS)
 - Verifica installazione (versione): `node -v`
- In una nuova directory
 - `hello.js`
 - Esecuzione: `node hello.js`
- `console.log()`
 - NodeJS: nello standard output del Sistema Operativo
 - Browser: nella console nei DevTools



```
let message = 'hello';  
console.log(message);
```

Variabili

- Per dichiarare una variabile si usa **var** (hoisting!) o **let** (ES 6)
 - case sensitive, myname è diverso da myName
- Non si esplicita il tipo, che può essere:

primitivi

- **string**: let name = 'Tim'; // apice singolo o doppio
- **number**: let value = 42; // non ci sono int, float, ...
- **boolean**: let flag = true; // o false
- **object**: let dog = { name : 'Zip', breed : 'Alsatian' };
 - **array**: let data = [1, 'Tom', false];

Notazione letterale

- Una variabile può cambiare tipo associato nel corso della sua vita
- L'operatore **typeof** ritorna la stringa che descrive il tipo dedotto da JS (o **undefined**)
 - Eccezione: null è di tipo null, ma typeof di null è "object"
- Per dichiarare costanti si usa **const** (ES 6)
 - const z = 42;

undefined vs null

Operatori aritmetici

- `+` addizione: `2 + 3`
- `-` sottrazione: `2 - 3`
- `*` moltiplicazione: `2 * 3`
- `/` divisione: `2 / 3`
- `%` modulo o resto: `2 % 3`
- `**` esponente: `2 ** 3` // **(ES 6)** sostituisce `Math.pow(2, 3)`
- `++` / `--` incremento / decremento (sia prefisso sia postfisso)

Operatori di assegnamento

- Operatori che assegnano alla variabile sulla sinistra ...
 - `=` il valore sulla destra
 - `+=` la somma dei valori a sinistra e destra
 - `-=` la differenza tra il valore di sinistra e quello di destra
 - `*=` il prodotto del valore di sinistra per quello di destra
 - `/=` la divisione del valore di sinistra per quello di destra

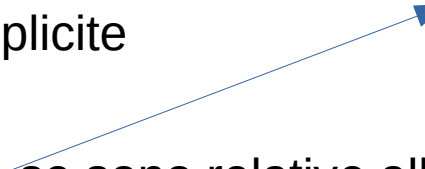
Operatori relazionali

- Operatori che ritornano un booleano
 - `===` stretta uguaglianza (stesso tipo e valore)
 - `!==` di stretta disuguaglianza (diverso tipo o valore)
 - `<` valore sulla sinistra è minore del valore sulla destra
 - `<=` minore o uguale
 - `>` il valore sulla sinistra è maggiore del valore sulla destra
 - `>=` maggiore o uguale
 - `!!` conversione a booleano, equivalente a `Boolean()`
- Gli operatori *non-strict* `==` e `!=` possono causare conversioni implicite



Truthy & falsy

- In contesto booleano, alcuni valori sono convertiti implicitamente a false, e sono detti “falsy”:
 - undefined, null, 0, -0, NaN, "" (la stringa vuota)
- Gli altri sono convertiti implicitamente a true, e sono detti “truthy”. Alcuni di questi sono:
 - [], {}, function(){} (ogni array, oggetto, funzione è truthy)
 - "false", "0" (solo la stringa vuota è falsy) → "0" == true, 0 == false, "false" == true
- Gli operatori **strict** === e !== non operano conversioni implicite
 - "42" == 42 (la stringa è convertita in numero) ma "42" !== 42
- Due reference a oggetti (array, funzioni) sono uguali solo se sono relative allo stesso oggetto (array, funzione)



```
let a = {};  
let b = {};  
  
a !== b
```


Operatori logici (e bitwise)

&&	AND
	OR
!	NOT
&	AND
	OR
^	XOR

```
let alpha = true;  
let beta = false;
```

```
console.log(alpha && beta);    // false  
console.log(alpha || beta);   // true  
console.log(!alpha);          // false  
console.log(alpha & beta);     // 0  
console.log(alpha | beta);     // 1
```

```
let gamma = 0b101;    // 5  
let delta = 0b110;    // 6
```

```
console.log(gamma & delta);    // 4 == 0100  
console.log(gamma | delta);    // 7 == 0111  
console.log(gamma ^ delta);    // 3 == 0011  
console.log(gamma && delta);    // 6
```

Stringa

- Una stringa è una sequenza **immutabile** di caratteri delimitata da apici singoli o doppi
- Per **concatenare stringhe** si usa il metodo **concat()** o l'operatore **+**
 - Conversione implicita da numero a stringa
`'Solution' + 42 === 'Solution42'`
- Conversione esplicita da numero a stringa via **toString()**
`a.toString() === '42' // se a === 42`
- Conversione esplicita da stringa a numero via **Number()**
`Number('42') === 42`

Lavorare con stringhe

- Lunghezza: `s.length`
- Accesso ai caratteri: `s[i]` // `i` in `[0, s.length-1]`
- Ricerca di sottostringa: `s.indexOf(sub)` // `-1` not found
- Estrazione di sottostringa:
 - `s.substring(beg, end)` // swap if `beg > end`
 - `s.slice(beg, end)` // end negativo == `len - end`
- Minuscolo: `s.toLowerCase()`
- Maiuscolo: `s.toUpperCase()`
- Modifica: `s.replace(sub, other)`
- Estrazione di componenti: `s.split(',')` // da stringa ad array

Array

- Collezione di oggetti di qualunque tipo
- Numero di elementi nella proprietà `length`
- Accesso agli elementi in lettura e scrittura `data[i]`
- Scansione di tutto l'array via for loop
- Da array a string via `join()`, `toString()`
- Per aggiungere un elemento: `push()`, `unshift()`
- Per eliminare elementi:
 - `pop()`, `shift()`
 - `splice()`
- *(altri metodi di Array più avanti)*

```
let data = [1, 'hello', [true, 42.24]];
console.log(data.length);
```

```
console.log(data[1], data[2][1]);
data[2] = false;
```

```
for(let i = 0; i < data.length; i++) {
  console.log(data[i]);
}
```

```
console.log(data.join(), data.toString());
```

```
data.pop();
data.shift();
data.push('push');
data.unshift('unshift');
```

Condizioni

- if – else if – else
 - se la condizione è vera, si esegue il blocco associato
 - altrimenti, se presente, si esegue il blocco “else”
- switch – case – default
 - Scelta multipla su valore
- Operatore ternario ?:
 - Ritorna la prima scelta se la condizione è vera, altrimenti la seconda

```
if (condition) {  
    doSomething();  
} else if (other) {  
    doOther();  
} else {  
    doAlternative();  
}
```

```
switch (value) {  
    case 1:  
        doOther();  
        break;  
    default:  
        doStuff();  
        break;  
}
```

```
let result = condition ? choice1 : choice2;
```

Loop

```
while (condition) {  
    // ...  
    if (something) {  
        condition = false;  
    }  
}
```

```
for (let i = 0; i < 5; i++) {  
    // ...  
    if (i == 2) {  
        continue;  
    }  
    // ...  
}
```

preferito

```
do {  
    // ...  
    if (something) {  
        condition = false;  
    }  
} while (condition);
```

insolito

forever

```
for (;;) {  
    // ...  
    if (something) {  
        break;  
    }  
    // ...  
}
```

Funzione

- Blocco di codice a cui è associato un nome, definite indicando
 - la keyword **function**
 - il nome (opzionale: funzioni anonime, notazione classica e “freccia grassa” **ES 6**)
 - una lista di parametri tra parentesi tonde
 - l'oggetto arguments, default per parametri **x = 0** (**ES 6**), parametro 'spread' **...va** (**ES 6**)
 - una lista di statement tra parentesi graffe
- In JavaScript sono oggetti, e dunque possono
 - essere assegnate a variabili, proprietà di oggetti, elementi di array
 - essere passate ad altre funzioni
 - contenere altre funzioni (metodi)
- Si invoca una funzione specificando
 - il suo nome
 - i valori da associare ai parametri – se non specificati, default o undefined

```
function f() {  
    console.log('hello');  
}
```

```
function g(a, b) {  
    return a + b;  
}
```

```
let f1 = function(a, b) {  
    return a + b;  
}
```

```
let f2 = (a, b) => a + b;
```

```
f();
```

```
let result = g(3, 5);
```

Oggetto

- Struttura, delimitata tra **parentesi graffe**, che contiene una **lista di proprietà** (attributi e metodi) separate da virgola
- Array associativo di proprietà definite come coppie **chiave-valore**
- Accesso proprietà per mezzo dell'operatore **.** o specificando il nome della proprietà fra parentesi quadre
- È possibile
 - aggiungere proprietà per assegnamento
 - rimuoverle via **delete**
- Uso della funzione **costruttore** per creare più oggetti
 - Non è permesso overloading
 - Ogni proprietà è inizializzata in uno statement
 - Uso della keyword **this**
 - Uso della keyword **new**

```
function Person(first, last) {  
    this.first = first;  
    this.last = last;  
  
    this.hello = () => 'Hello from ' + p.first;  
}  
  
let p = new Person('Tom', 'Jones');
```


Oggetto /2

- Struttura definita dal *prototype* del suo costruttore
- Ereditarietà via prototipo
- Object è alla base della gerarchia
- La relazione può essere indicata nel costruttore 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);  
    /* ... */  
}
```

Math

Costanti e funzioni matematiche di uso comune

- `Math.E`, `Math.PI`, `Math.SQRT2`, ...
- `Math.abs()`
- `Math.ceil()`, `Math.floor()`
- `Math.cos()`, `Math.sin()`, `Math.tan()`, ...
- `Math.exp()`, `Math.pow()`, `Math.sqrt()`, ...
- `Math.max()`, `Math.min()`

Date

- Data + ora fino al secondo
 - `new Date()`
 - `new Date(2019, 10, 15, 20, 58, 51)`
 - `new Date("15 October 2019 12:23:15")`
- Differenza: millisecondi tra due date
- Getter e setter per leggere o modificare componenti
 - `getDate()`, `setDate()`, ...

Eccezioni

- Gestione 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
```



operatore rest

```
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)
- Ordine in-place: `sort()`
- Copia di intervallo: `slice()`
- Inversione in-place: `reverse()`
- Copia filtrata: `filter()`
- Copia di array/iterable: `Array.from()` (ES 6)

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

// ...
array.sort(
  (left, right) => left == right ? 0 :
    left < right ? -1 : 1);

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

array.reverse();

let odds = array.filter(value => value % 2);

let chars = Array.from('hello');
```

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)

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

for ... of
(iterabili)

ES 6

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

Array.forEach()

```
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

Un solo ctor!

```
class Person {  
  constructor(first, last) {  
    this.first = first;  
    this.last = last;  
  }  
  
  fullInfo() {  
    return this.first + ' ' + this.last;  
  }  
}
```

ES6

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

Pseudoproprietà: get e set

```
class Person {  
  // ...  
  
  get fullName() {  
    return this.first + ' ' + this.last;  
  }  
  
  set fullName(name) {  
    let buffer = name.split(' ');  
    this.first = buffer[0];  
    this.last = buffer[1];  
  }  
}
```

ES6

```
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

ES6

```
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());
```