

JavaScript

- Le basi del linguaggio
 - Nato nel browser
 - Ora anche uso indipendente: Node JS
- Progetto di riferimento
 - <https://github.com/egalli64/nesp> (*modulo 3a*)
 - Node.js
 - VS Code

JavaScript

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

Node JS

- Piattaforma per server app in JavaScript
 - Ben supportata da VS Code
- <https://nodejs.org/en/download/> (LTS)
- In una nuova directory
 - hello.js
 - Esecuzione: node hello.js
- console.log()
 - NodeJS: nello standard output del Sistema Operativo
 - Browser: nella console nei DevTools
- Esecuzione in Code: F5 (debug) o Ctrl+F5
 - Se non Code non associa automaticamente JS a Node → Settings (Ctrl+,)



hello.js

```
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 della variabile, il cui valore può essere di tipo:
 - **string**: let name = 'Tim'; // apice singolo o doppio
 - **number**: let value = 42; // doppia precisione IEEE 754
 - **boolean**: let flag = true; // o false
 - **object**: let dog = { name : 'Zip', breed : 'Alsatian' }; // notazione letterale
 - **Array**: let data = [...]; // tra gli altri object: **Function**, Date, RegExp, Error, ...
- L'operatore **typeof** ritorna la stringa che descrive il tipo del valore (*o undefined*)
 - Eccezioni: null è di tipo null, ma typeof è "object"; le funzioni sono "object" ma typeof è "function"
- Per dichiarare costanti si usa **const** (ES 6)
 - const z = 42;

primitivi
scalari

undefined: solo dichiarata
null: non c'è un "buon" valore

Operatori aritmetici

- addizione: $2 + 3$
 - $0.1 + 0.2 = 0.30000000000000004$
- sottrazione: $2 - 3$
- moltiplicazione: $2 * 3$
- divisione: $2 / 3 = 0.6666666666666666$
- modulo o resto: $2 \% 3 = 2$
 - $2.3 \% 2.1 = 0.199999999999999973$
- esponente: $2 ** 3$ // (ES 6) affianca `Math.pow(2, 3)`
- `++` e `--` incremento / decremento (sia prefisso sia postfixo)

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

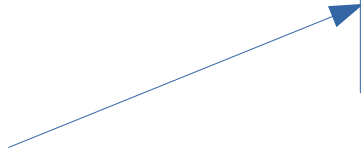
Operatori relazionali

- 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 → **falsy**:
 - undefined, null, 0, -0, "" (la stringa vuota)
 - **NaN**, “not a number”, indica un risultato numerico non valido
- Gli altri sono convertiti implicitamente a true, e sono detti **truthy**. Ad esempio:
 - [], {}, 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, o NaN) ma "42" !== 42
- Due reference a oggetti sono uguali solo se sono relative allo stesso oggetto
 - Lo stesso vale per array e funzioni
- Conversione esplicita a booleano, raramente utilizzata, via Boolean()



```
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
```

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;  
    }  
    // ...  
}
```

Oggetto

- Struttura, delimitata tra **parentesi graffe**, contiene una **lista di proprietà** separate da virgola
- Array associativo di proprietà definite come coppie **chiave-valore**
 - La chiave è una stringa, il valore può essere un dato di qualunque tipo o un metodo
- Accesso alle proprietà
 - via operatore di dereferenziazione “.” o specificando il nome della proprietà fra parentesi quadre
- È possibile aggiungere proprietà per assegnamento e rimuoverle via **delete**
- Uso della **funzione costruttore**
 - Non è permesso overloading
 - Ogni proprietà è inizializzata in uno statement
 - Uso delle keyword **this** e **new**
- **Object.create(obj)**
 - Creazione di un oggetto via prototipo

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

Funzione

- Blocco di codice a cui è associato un nome, definita indicando
 - la keyword **function**
 - il nome (opzionale: funzioni anonime)
 - tra parentesi tonde, una lista di **parametri**
 - tra parentesi graffe, una lista di statement
- 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 specificandone il nome e argomenti
 - Passaggio ai parametri **by-value** (per oggetti è il reference – come in Java)
 - Se non si passa un argomento atteso dalla funzione, il parametro è *undefined*
 - Se si passa un argomento non atteso, non viene considerato

```
function f() {  
    console.log('hello');  
}  
  
function g(a, b) {  
    return a + b;  
}  
  
let ga = function(a, b) {  
    return a + b;  
}
```

Stringa

- Sequenza **immutabile** di caratteri
 - Delimitata da apici singoli, doppi, o backtick (` Alt+96)
 - Placeholder in **template literal** (ES 6) via `${ }`
 - Concatenazione via metodo **concat()** o operatore **+**
 - Conversione implicita da numero a stringa
- Conversioni esplicite
 - da numero a stringa via metodo **toString()**
 - da stringa a numero via **Number()**
 - costruttore del wrapper di number usato come funzione
 - NaN in caso di errore
 - da stringa a numero intero via **parseInt()**
 - NaN in caso di errore

```
let a = 'the ';  
let b = "answer ";  
let c = `is `;  
let d = 42;  
  
let e = a + b + c + d;
```

```
d.toString() === '42'  
  
Number('42.7') === 42.7  
  
parseInt('42') === 42)  
  
isNaN(parseInt('hello'))
```

Lavorare con stringhe

- Lunghezza: nella proprietà **length**
- Accesso ai caratteri con l'operatore **[]** o metodo **charAt()** // indice in [0, s.length-1]
- Ricerca di sottostringa: **s.indexOf(sub)** // -1 not found
 - E anche: **startsWith()**, **endsWith()**, **includes()**
- Estrazione di sottostringa:
 - **s.slice(beg, end)** // end negativo == len - end
 - **s.substring(beg, end)** // swap if beg > end
- Minuscolo: **s.toLowerCase()**
- Maiuscolo: **s.toUpperCase()**
- Modifica: **s.replace(sub, other)**
- Estrazione di componenti: **s.split(',')** // da stringa ad array

Array

- Oggetto che *assomiglia* a un array
 - Proprietà **length**: indice massimo più uno
- Accesso agli elementi in lettura e scrittura con operatore **[]**
 - In lettura ritorna *undefined* se non esiste la chiave indicata
 - In scrittura può creare buchi nell'array
- Scansione di tutto l'array via for loop (*buchi inclusi*)
- Da array a string via metodi **join()**, **toString()**
- Per aggiungere un elemento: **push()**, **unshift()**
- Per eliminare elementi: **pop()**, **shift()**
- **splice()**
 - Da un dato indice, elimina *n* elementi, ne inserisce altri(*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');
```


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()`, ...