Creative Web Development

Guarise, Degl'Innocenti, Rossi - 2018

Lezione 3

A cura di: **Prof. Degl'Innocenti**

Teoria

I framework

- **Software di supporto su cui sviluppare** un proprio progetto
- Architettura già pronta da cui partire
- Rappresenta la **struttura** di una parte o di tutto il progetto
- Di solito implementa un **pattern architetturale**
- **Facilita** lo sviluppatore
- Rende disponibili delle **classi astratte da implementare**
- Rende disponibili delle **librerie da utilizzare**
- A volte corredato da **tools di sviluppo appositi**: IDE, debugger

Esempi di framework JS

- **Angular.js**
- **Angular2**
- **Socket.io**
- **Ionic**
- **Node**

Ma anche, in JS: jQuery, Ember, Vue.js
Altri: https://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks

- In **Php**: Symphony, Laravel, PhpCake
- In **Python**: Django
- In **Ruby**: RubyOnRails

I tools

- Strumenti per **migliorare la metodologia di lavoro** e sviluppo
- Un tool può **gestire un determinato compito** in una delle fasi di sviluppo
  - Versionamento del Software
  - Ambiente di sviluppo
  - Console di output
  - Strumenti di analisi e debugging
  - Strumenti di testing
  - Strumenti di distribuzione del software

Tools: Versionamento del lavoro in team

- Con i tool di versionamento o di **controllo versione** possiamo assegnare un numero (o versione) ad ogni nuova modifica del nostro codice
- Mantiene in memoria **tutte le modifiche effettuate**
- Creare un “albero” delle modifiche e **spostarci avanti e indietro** senza perdere il lavoro svolto
- Permette di creare dei rami (**branch**) di sviluppo separati a partire da una versione comune, e in certi casi di riunirli in seguito
- Permette a più sviluppatori di **lavorare sullo stesso repository di codice in modo collaborativo**
- https://git-scm.com/book/it/v1/Per-Iniziare-Il-Controllo-di-Versione

Tools: Versionamento del lavoro in team

- Strumenti di controllo versione:
  - GIT
  - Mercurial
  - SVN
- **Tutarial interattivo GIT online:** https://try.github.io/levels/1/challenges/1
- E’ possibile utilizzare strumenti ad interfaccia grafica invece che a linea di comando: **SourceTree** https://www.sourcetreeapp.com/
- E’ possibile hostare online i propri repository di codice su servizi gratis:
  - GitHub: gratis per repository aperti a tutti (con vasta community)
  - BitBucket: gratis anche per repository privati

Tools: IDE

- **Ambiente di sviluppo integrato**
- Aiuta a scrivere il codice all'interno di un ambiente di sviluppo
- Aiuta a rilevare errori già in fase di scrittura (LINT)
- Contiene strumenti per sviluppo e debugging
  - Compilatore e/o interprete
  - Controllo di versione integrato
  - Connessione a database
  - Installazione librerie
  - Strumenti di analisi per i linguaggi orientati agli oggetti

Tools: Esempi di IDE

- Per il JS:
  - Visual Studio Code
  - Sublime
  - NetBeans
  - WebStorm
  - Eclipse

- Per altri linguaggi:
  - Android Studio
  - XCode
  - IntelliJ Idea Suite
  - ...

Tools: IDE, Visual Studio Code

- Per Windows, Mac e Linux
- Supporta molti linguaggi
- Gratuito
- Funzionalità espandibili attraverso moduli di estensioni
- Supporta il lancio di istruzioni a linea di comando direttamente dall'interno dell'editor
- https://code.visualstudio.com/

Tools: La console CLI

- Interfaccia a linea di comando
- Interfaccia testuale tra utente ed elaboratore
- Permette l'esecuzione di comandi o istruzioni all'interno del workspace dell'ambiente e lettura dei risultati di elaborazione
- Specifica per un determinato linguaggio (JS, Bash, Batch, vari REPL)
- REPL: Read–eval–print loop
- Esempi:
  - Chrome Web Development Tool (Ispeziona elemento di Chrome e sua console)
  - Terminale di Mac (Bash)
  - Prompt dei comandi di Windows (Batch)
  - Vari tools online per il JS: https://repl.it/repls/FrillySpryMyna

L'esecuzione

- Comando di **Run**
- Per linguaggi **interpretati** → Esegue il codice nell'interprete (es. JS, PHP)
- Per linguaggi **compilati** → Compila il codice e lo esegue (es. Java, C, Ruby)
- La reale fase di esecuzione del codice
- Può essere eseguita da:
  - Console
  - Browser
  - Un programma
  - Un App
  - Sistema Operativo
  - Macchine Virtuali
  - ... in generale qualsiasi elaboratore

Il testing

- Fase di esecuzione subito dopo la scrittura del codice in cui lo sviluppatore o chi per lui esegue il codice e ne **verifica il corretto funzionamento**
- Può avvenire **manualmente** verificando che a determinati input corrisponda il corretto output
- Può avvenire in **Unit Testing**, ovvero tramite la creazione un apposito moduli di codice che altro non fa che passare degli input e aspettarsi degli output, e restituisce successo o fallimento a seconda del risultato (mocha, chai, karma)
- **TDD: Test Driven Development** → Modalità di sviluppo che antepone la creazione di un modulo di test allo sviluppo del codice concreto che esegue le operazioni
- **Debugger**: funzionalità che permette l'esecuzione del codice step by step con visuale del valore delle variabili in un dato momento dell'esecuzione

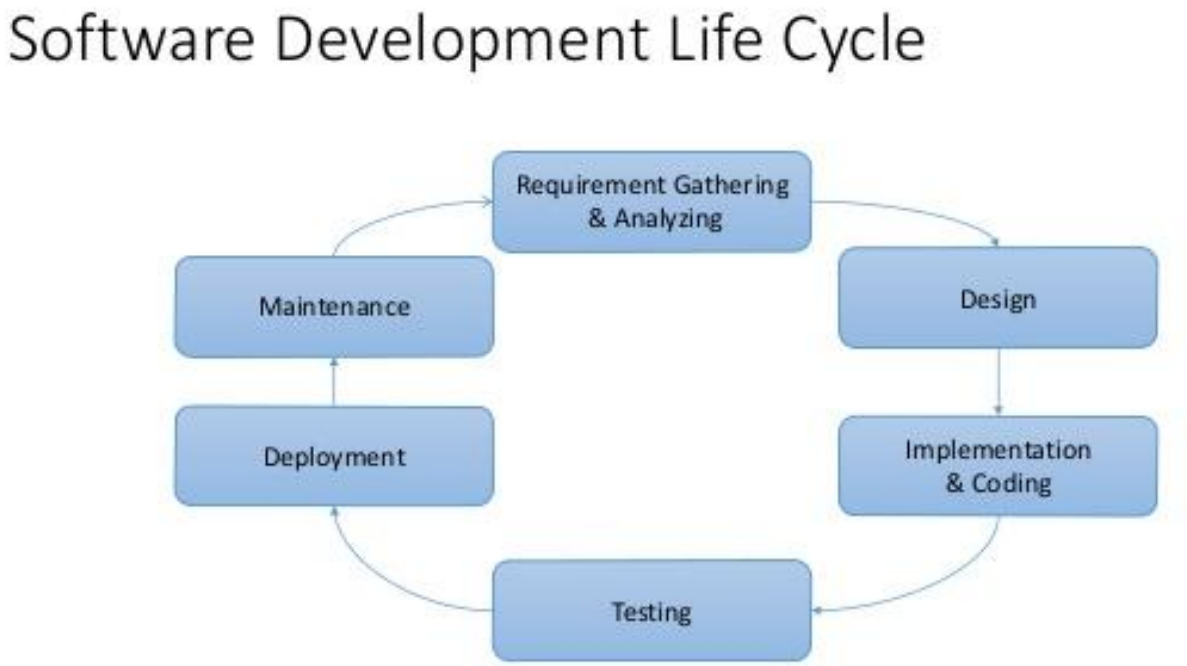
Il building, il deploying e l'ambiente di produzione

- **Build**: trasformazione del codice in un artefatto concreto che può andare in run sulla macchina
  - Es. : trasformazione di un progetto Ionic in un software eseguibile da smartphone
  - Tramite compilazione o assemblaggio **in un unico pacchetto standalone**
- **Deploy**: consegna o rilascio al cliente, con relativa installazione e messa in funzione o esercizio sull'ambiente di destinazione
- **Ambiente di produzione:** Ambiente di finale utilizzo del software da parte di tutti gli utenti: ove risiede per la suo fruizione finale, quando in produzione il software
- **Altri ambienti**: Dev, Test, Staging

Manutenzione e sviluppo

- Una volta in **produzione**, il software è utilizzato da tutti gli utenti
- In caso sia necessario effettuare modifiche o fix, lo sviluppatore procede sul suo ambiente di **dev**
- Una volta ultimate e testate le modifiche, esegue un push delle modifiche su **test o staging**, dove il QA o il project manager verifica la correttezza dei nuovi o modificati comportamenti del programma
- Una volta verificato con successo, il codice può essere portato in **produzione**
- *... e così via*

Ciclo di sviluppo del software



Pratica

Le variabili

- In JS una variabile contiene un determinato **tipo di dato**
  - numero interno
  - numero con virgola
  - booleano
  - Stringa
  - vettore (array)
  - oggetto
  - null, undefined
- Una **locazione di memoria** può essere dichiarata in diversi modi
  - Variabile
  - Costante
  - Variabile con visibilità limitata

I tipi di variabile

- numero interno
- numero con virgola
- booleano
- Stringa
- vettore (array)
- oggetto
- null, undefined

Variabili: Numero intero

- Numeri interi: 127
- Numeri negativi: -64

```
var number = 127;

var number = -64;

var number = 0;
```

Variabili: Numero con virgola

- Si utilizza il **punto** • non la virgola (che serve invece a separare liste di dati)
- 0.99
- 15.4
- -100.32

```
var number = 0.99;

var number = 15.4;

var number = -100.32;
```

Variabili: Booleano

- Rappresenta uno stato VERO / FALSO
- true
- false

```
var boolean = true;

var boolean = false;

var isNetworkAvalaible = true;
```

Variabili: Stringa

- Rappresenta una serie di caratteri testuali
- Il valore è contenuto tra singoli apici '...' o doppi apici "..."

```
var string = 'una stringa di testo';
var string = 'testo';
var string = 'altro testo';
```

Variabili: Array

- Array o vettore
- Contiene una lista di valori racchiusi tra parentesi quadre **[ ]**
- e separata da virgola **,**

```
var array = [ 1, 2, 3, 10, 15, 20];

var array = [ -1, 2.5, 15, 20];

var array = [ true, false, false];

var array = ["Stringa 1", "Stringa 2", "Stringa 3"];
```

Vedremo in seguito in modo più approfondito.

Variabili: Oggetto

- Contiene delle proprietà (valori) associati a delle etichette (label)
- Può contenere valori di vario tipo: variabili, funzioni, ecc...

```
var object = {   prop: "value",
                 prop2: 10,
                 prop3:    function () {
                               console.log("Testo");
                           },
                 prop4: -10
             };
```

Vedremo in seguito in modo più approfondito.

Varibili: speciali

- **null** ➜ identifica un valore speciale per una variabile che non contiente alcun valore al suo interno
- **undefined** ➜ identifica un valore speciale per una variabile che non è ancora stata inizializzata
- **NaN** ➜ Risultato di operazioni aritmetiche impossibili (es. 0/0)

```
var variable = null;

var varibale = undefined;

var variable = NaN;
```

Esercitazione

1. Assegna, modifica il valore e stampa di una variabile intera
2. Assegna, modifica il valore e stampa di una variabile con virgola
3. Assegna, modifica il valore e stampa di una variabile Booleana
4. Assegna, modifica il valore e stampa di una variabile Stringa
5. Assegna, modifica il valore e stampa di una variabile Array
6. Assegna, modifica il valore e stampa di una variabile Oggetto
7. Assegna, modifica il valore e stampa di una variabile Null
8. Assegna, modifica il valore e stampa di una variabile Undefined

Dichiarazione delle variabili

- Una **locazione di memoria** può essere dichiarata in diversi modi
  - Variabile *var*
  - Costante *const*
  - Variabile con visibilità limitata *let*

Dichiarazione: var

- Dichiara una variabile all'interno dello scope di uno script o funzione
- Una volta creata può essere utilizzata all'interno del suo scope
- Una volta creata può essere ri-assegnata modificando il tipo di valore contenuto
- La dichiarazione più utilizzata in plain JS

```
var outsideVariable = 10;
outsideVariable = 7;
function myFunction(){
   var insideVariabile = 5;
}
console.log(outsideVariable); //stampa 7
console.log(insideVariable); //dà errore ReferenceError
```

Dichiarazione const

- Dichiara un allocazione di memoria immutabile
- si scrive con tutte lettere maiuscole e underscore _
- Una volta assegnata non si può più scrivere all'interno ma solo leggere
- Utile per memorizzare valori immutabili

```
const MY_CONSTANT = 10;

console.log(MY_CONSTANT); //stampa 10
MY_CONSTANT = 7;
//^^^ dà errore: TypeError: Assignment to constant variable.
```

Dichiarazione: let

- Dichiara un'allocazione valida solo all'interno di un **blocco di codice**
- Può essere ri-dichiarata all'interno dello stesso blocco di codice

```
let a = 10;
if( a == 10 ){
    let b = 5;
    console.log(b); //stampa 5
}
console.log(a); //stampa 10
console.log(b); //dà errore: ReferenceError
```

Esercitazione

1. Crea una variabile con *var*, modificane il valore e stampala
2. Crea una variabile costante *const* e stampala
3. Crea una variabile costante *const*, prova a modificarne il valore e ottieni l'errore derivante
4. Crea una variabile con *let* dentro un blocco if a piacimento, e stampala dentro il blocco stesso
5. Crea una variabile con *let* dentro un blocco if a piacimento, prova a stamparla fuori dal blocco e ottieni l'errore derivante

Gli operatori

- Operazione matematica, aritmetica o logica che operando su uno o più valore restituisce un risultato di vario tipo (numero, booleano, altro)
- Aritmetici: `+, -, x, /, %, ++, --`
- Relazionali: `<, >, <=, >=, ==, !=, ===, !==`
- Logici: `&&, ||, !`
- Su Strighe: `+`
- Operatori di casting

Operatori: Aritmetici

- **Per eseguire operazioni aritmetiche su numeri e numeri con la virgola:**
- `+` addizione
- `-` sottrazione
- `/` divisione
- `*` moltiplicazione
- `%` modulo o resto

```
var number = 5 + 10;
number = 7 - 10;
number = 2 + 3 * 7;
number = ( 2 + 3 ) * 7;
```

Operatori: Aritmetici Unari

- Operano su di una variabile per incrementare o decrementare il valore di una sola unità

```
var number = 10;
number++;
console.log(number); //stampa 11

var number = 10;
number--;
console.log(number); //stampa 9
```

Operatori: relazionali

- Operano su valori numerici e restituiscono un booleano
- Spesso utilizzati nelle condizioni dei costrutti IF, FOR, WHILE

- `<` minore
- `<=` minore o uguale
- `>` maggiore
- `>=` maggiore o uguale
- `==` uguale
- `!=` diverso
- `===` strettamente uguale
- `!==` strettamente diverso

Operatori: relazionali

```
var number = 10;
var number2 = 11;
if( number < number2 ){ doSomething(); }

var number = 10;
var number2 = "10";
console.log(number == number2); //stampa true

var number = 10;
var number2 = "10";
console.log(number === number2); //stampa false
```

Operatori: Logici

- **Operano su valori Booleani e restituiscono un booleano**
- && ➜ and : restituisce vero se entrambi le condizioni sono vere
- || ➜ or : restituisce vero se une delle due condizioni è vera
- ! ➜ not : restituisce il valore negato (true ➜ false ; false ➜ true)

```
var bool = true;
var bool2 = false;
console.log(bool && bool2); //stampa false
console.log(bool || bool2); //stampa true
console.log(!bool); //stampa false
```

Operatori su: stringhe

- **Per manipolare stringhe: concatenazione, riduzione, ricerca, sostituzione...**
- **+** ➜ permette di concatenare due stringhe

```
var string = "ciao ";
var string2 = "mondo";
console.log(string + string2); //stampa 'ciao mondo'
```

- L'oggetto stringa propone anche tanti altri metodi per le più svariate funzioni:
  - lunghezza della stringa, trovare posizione di un dato carattere, riduzione della lunghezza...

Altri metodi: https://www.w3schools.com/js/js_string_methods.asp

Operatori: Cast

- Il cast è quella modalità che permette di trasformare una variabile di un dato tipo in un altro tipo, mantenendo la coerenza del dato
- **Il cast è eseguito automaticamente** da JS in vari casi:
  - **Per gli operatori relazionali *Loosy*** (non stretti) ➜ `==` , `!=`
  - In caso di concatenazione a stringa
  - Nelle operazioni matematiche (per es, da intero a con la virgola)
- Usando il cast, possiamo trasformare il tipo di variabile
  - numero ➜ stringa
  - stringa ➜ numero
- **parseInt()**
- **parseFloat()**
- **""+number**

Operatori: Cast

```
var string = "10";
var number = parseInt(string);
console.log(string); // "10"
console.log(number); // 10


var string = "10.5";
var number = parseFloat(string);
console.log(string); // "10.5"
console.log(number); // 10.5
```

Esercizi

1. Leggi un valore stringa e tramite **cast** trasformalo in numero
2. Leggi due valori numerici, fai un operazione e stampa il risultato
    a. operazione di addizione
    b. sottrazione
    c. moltiplicazione
    d. divisione
3. Leggi due valori numerici e crea un IF/ELSE che stampi un messaggio
    a. Se il primo numero è maggiore del secondo
    b. Se il primo numero è minore o uguale del secondo
    c. Se il primo numero è diverso dal secondo
4. Crea due variabili boolean, una true ed una false, e crea un IF/ELSE che stampi un messaggio
    a. Se sono entrambi veri
    b. Se sono uno dei due è vero
    c. Se sono entrambi falsi
5. Leggi due stringhe e stampane la concatenazione utilizzando l'operatore +