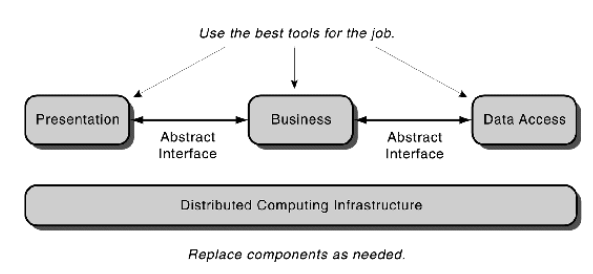
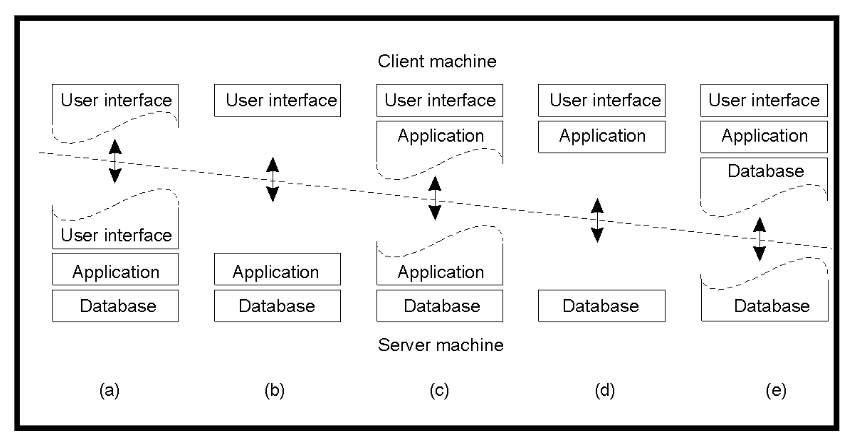
# Web App - AJAX

Il browser = rendering delle pagine HTML.

### ARCHITETTURE MULTI TIER



Ogni componente può essere assegnato a una figura diversa. I componenti possiedono tutti un’interfaccia astratta.



Nel caso A il server fa tutto, mentre la parte di client fa solo la parte di presentazione.

Es. in un videogioco, il client cerca di elaborare più dati possibili, come il caso C. Nel caso D il client dal server prende solamente i dati, mentre nel caso e addirittura fornisce una parte di essi al server.

### 

Per poter implementare gli altri casi delle architetture multilivello (tranne A) il client deve essere programmabile.

**AJAX** è una tecnica che aiuta a sviluppare pagine web dinamiche.

* Validazione dei dati in tempo reale: per esempio se i dati del form sono sintatticamente corretti
* Auto completamento
* Controlli dell UI

I componenti di AJAX sono:

* HTML e CSS: presentazione di informazioni
* DOM
* XML, JSON: invio dei dati
* XMLHttpRequest: scambio di dati tra client e server asincrono
* JavaScript

**JAVASCRIPT** è un linguaggio di scripting a oggetti, debolmente tipizzato, inizialmente pensato per l’esecuzione di semplici script. Permette anche lui di creare pagine HTML dinamiche. Permette di effettuare le richieste HTTP al server in modo trasparente rispetto all’utente. Può richiedere dati in formato testo puro e XML.

AJAX usa *anche* JavaScript per effettuare chiamate asincrone.

La differenza tra un’architettura web tradizionale e una dinamica è che in quella dinamica c’è un costante scambio di msg, causato da AJAX. Quella dinamica è basata su un sistema asincrono per lo scambio dei messaggi.

User Interface <-> AJAX Engine <-> Web Server

Nel client sono presenti User Interface e AJAX Engine.

Uso AJAX =

* API e formati interscambio
* eseguire ciclicamente funzioni javascript

JSON > XML perchè è simile a javascript

### STRUTTURA DELLA PAGINA

JavaScript non possiede nessuna funzione built-in per la stampa a video.

**JavaScript può visualizzare i dati in diversi modi:**

* Scrivendo in un elemento HTML utilizzando `innerHTML`:
  + document.getElementById("demo").innerHTML = "Hello world!";
  + Questo sostituisce il contenuto di un elemento HTML con un identificatore.
* Mostrando un alert box con `window.alert()`:
  + window.alert("Hello world!")
  + Questo apre una nuova finestra (alert box) con il contenuto indicato.
* Scrivendo sulla console del browser con `console.log()`:
  + console.log("Letti i primi " + count + " valori.");
  + Questo scrive messaggi di debugging sulla console per il controllo dell'esecuzione.
* Sostituendo l'intera pagina con `document.write()`:
  + document.write("Hello World");
  + Questo metodo è sconsigliato perché sostituisce l'intera pagina.

JavaScript non ha funzioni integrate per la stampa o la visualizzazione dei dati. Deve utilizzare metodi come questi per interagire con il DOM HTML e mostrare i risultati all'utente.

**Controllo e input dati**:

* L'invocazione di funzioni avviene quando qualcosa le richiama.
* Gli eventi JavaScript reagiscono agli eventi HTML che accadono sugli elementi HTML.
* È possibile associare gestori di eventi agli elementi del DOM HTML che generano eventi utilizzando addEventListener.

Ecco un esempio di codice che mostra come visualizzare la data quando un pulsante viene cliccato:

|  |
| --- |
| <!DOCTYPE html> <**html**> <**body**>  <**p**>Clicca sul pulsante per visualizzare la data.</**p**>  <**button** id="button">Che ore sono?</**button**>  <**p** id="demo"></**p**>  <**script**>  document.getElementById('button').addEventListener('click', displayDate);  **function** **displayDate**() {  document.getElementById("demo").innerHTML = Date();  }  </**script**> </**body**> </**html**> |

In questo esempio, quando il pulsante viene cliccato, la funzione displayDate viene chiamata e visualizza la data corrente nell'elemento HTML con l'id "demo". Date() è una classe.

**PARADIGMA A EVENTI**

L’applicazione è “reattiva”. Non è possibile identificare staticamente un flusso di controllo unitario. Il paradigma principale inizializza l’applicazione, istanzia gli osservatori, assicurandogli gli opportuni handler. L’associazione può essere dinamica.

Si può associare uno o più gestori di eventi ad ogni elemento del DOM HTML che genera eventi. Si usa AddEventListener (element.addEventListener(event,function)). L’evento può essere per esempio “click”, mentre la funzione deve essere ciò che viene richiamata quando accade l’evento.

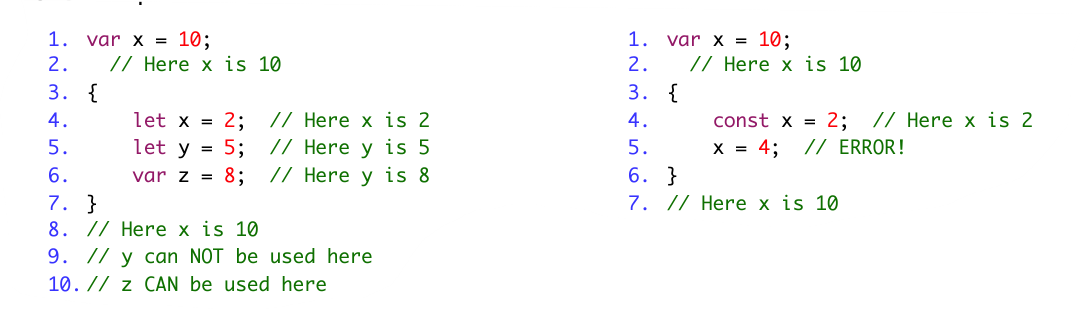
*poi spiega come funziona javascript ma a noi non ci interessa (forse, non è che dobbiamo saper motivare le parti di codice?)*

### PROGRAMMAZIONE IN JAVASCRIPT

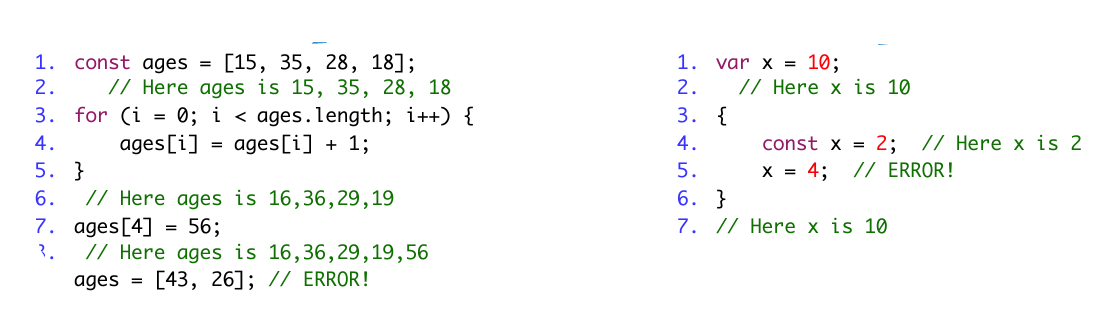
let = variabile che vale solo nel blocco in cui è dichiarata (le const sono delle let, ma se sono array o var. composte possono essere cmq modificate)

var = variabile globale normale

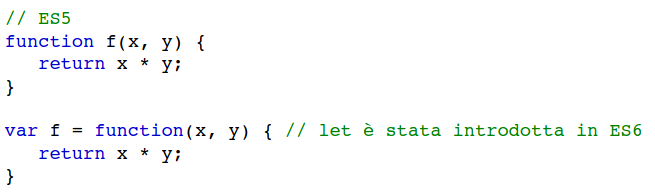
Una buona regola è usare var solo per le variabili globali, const e let in tutti gli altri casi



Le const sono contenitori costanti, ma i valori contenuti possono cambiare.Le variabili semplici sono immodificabili, mentre i valori delle variabili composte possono cambiare.

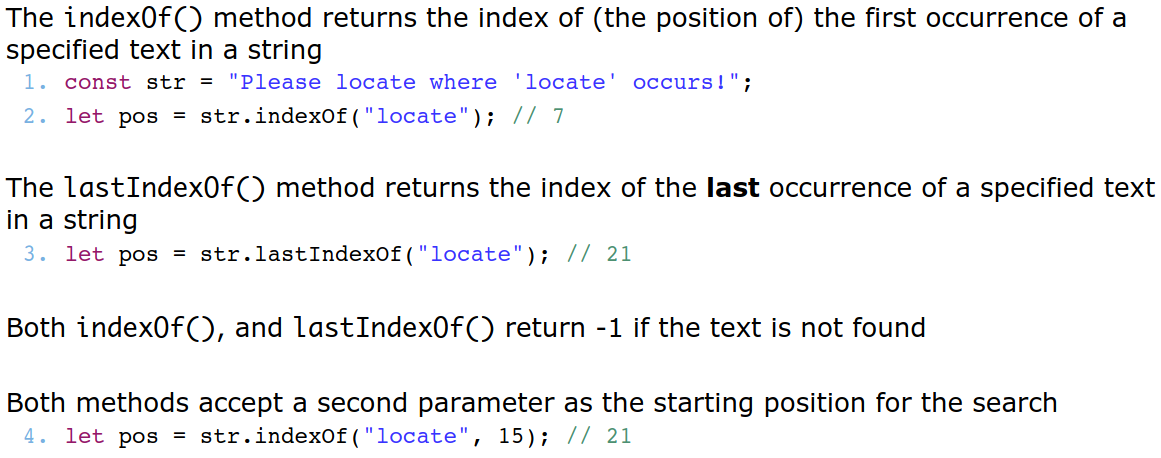


Esistono due modi per definire le funzioni:





La gestione delle stringhe è uguale a Java.



Per manipolare gli array:

* array.push(“elemento”)
* array.push() = restituisce la lenght
* array.pop()
* array.shift() = rimuove il primo elemento di un array e lo rimette a posto
* array.unshift(“elemento”) = mette un elemento nella prima posizione dell’array e rimette a posto l’array
* array.slice(3, 0, “ciao”, “bro”) = nella posizione 3 inserisci “ciao” e “bro” e elimina 0 elementi

Per accedere ai valori di un array:

for (let x **of** numbers) {sum = sum + x;}

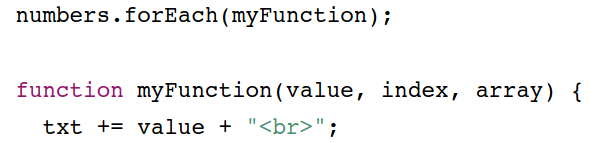
La versione di JS ES6 ha introdotto le classi che servono per definire dei prototipi di oggetti per dichiarare più entità oggetto dello stesso tipo.

Tutti i valori in JS, tranne i valori dei tipi primitivi, sono oggetti.

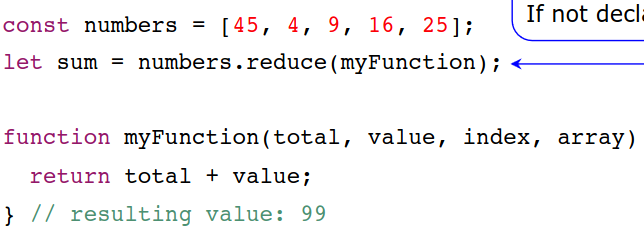
Se voglio manipolare un oggetto uso -> for (let x **in** numbers) {sum = sum + x;}

ALtri elementi di programmazione sono:

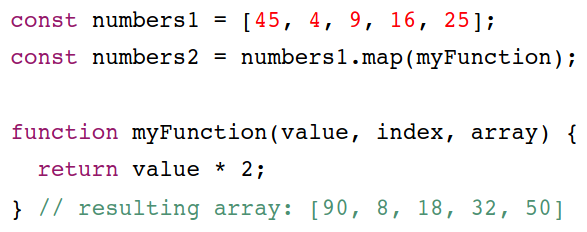
**forEach** che cicla un array applicando ad ogni iterazione il corpo di una funzione



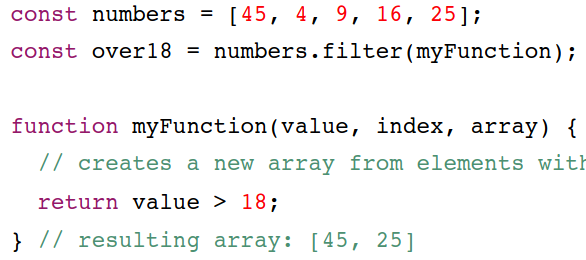
**reduce** cheriduce un array ad un solo valore (somma o sottrazione). Restituisce un nuovo array di 1 solo elemento.



**map** che crea un nuovo array a partire da un altro



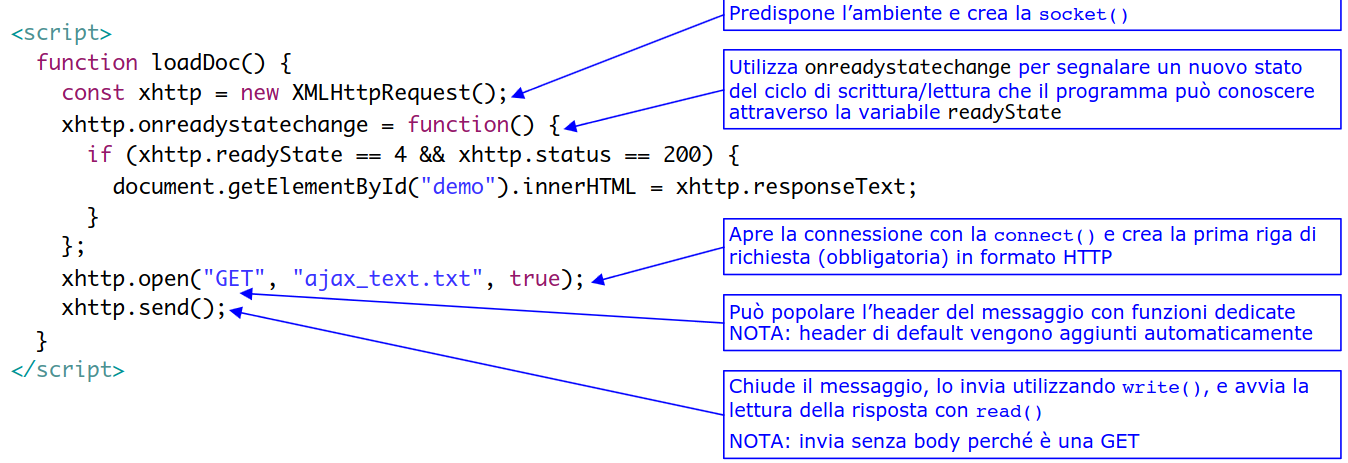
**filter** che crea un nuovo array con gli elementi che passano il filtro



# Interazioni con i Server

Funzione loadDoc(): durante una comunicazione asincrona il server predispone una callback, cosicché quando lo stato del server cambia, la funzione viene eseguita e risponde al client. Il client risponde 200 OK con msg corretto (stato=4).

Quando si dice che AJAX effettua chiamate asincrone si intende questo:



*Ricordarsi che durante una send non è necessario specificare la grandezza del messaggio.*

**XMLHttpRequest()** definisce funzioni che nascondono le chiamate alle API di Sistema. Infatti è supportato da tutti i browser. Possiede 3 proprietà:

**readyState** rappresenta lo stato della XMLHttpRequest (chiusa, aperta, loading…). Viene chiamata dalla **onreadystatechange**, che scatta quando la risposta del server arriva ed è pronta ad essere processata. Invece lo **status** restituisce lo stato (200 OK, 404 …).

### OGGETTI JSON

|  |
| --- |
| {  "name": "John Doe",  "age": 30,  "isStudent": **false**,  "Array": ["Math", "Science", "History"],  "OggettoAddress": {  "street": "123 Main St",  "city": "Anytown",  "postalCode": "12345"  } } |

**parse()** chiamato da un oggetto JSON, crea l’oggetto java corrispondente

**stringify()** ritorna una stringa JSON in base a certi valori passati

Limiti di JSON:

* Non supporta i commenti
* Rigidità di Tipo: JSON ha una gamma limitata di tipi di dati (stringa, numero, oggetto, array, boolean, null), il che può richiedere soluzioni alternative per rappresentare dati più complessi.

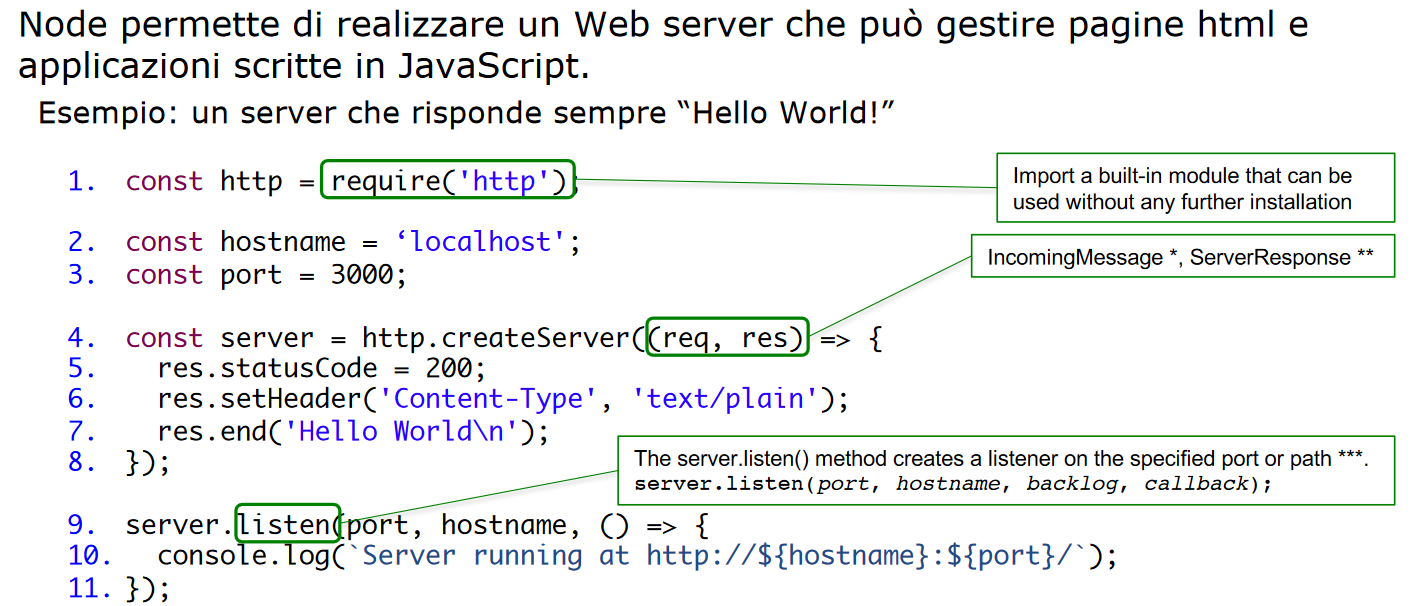
# Server-side Programming

### Node.js

Node.js è una piattaforma basata su javascript, che permette di scrivere applicazioni web. Occupa poche risorse. Il gestore dei moduli (NPM) permette di mantenere aggiornate le librerie usate, che sono anche open-source.

**Approccio asincrono**

Significa che invece di aspettare che l'operazione venga completata prima di passare alla successiva, Node.js può iniziare una nuova operazione mentre quella precedente è ancora in corso, creando una funzione di callback che che completi l’effetto desiderato.



Listen() trasforma la socket in uno affidabile e annuncia che è pronta a ricevere.

L'approccio asincrono di Node.js è ideale per chi richiede molte operazioni I/O, come server web, API, etc.



È la D. Qualsiasi richiesta a "<http://localhost:8081/>" attiverà la funzione che legge e restituisce "provaGet.html"

### MODELLO A EVENTI

**Event Loop**: Node.js utilizza un Single Event Loop (SEL). Il SEL è un ciclo continuo che ascolta eventi e li gestisce man mano che si verificano. Questo loop gestisce operazioni I/O in modo non bloccante.

**Single Thread**: Anche se Node.js usa un singolo thread per eseguire il codice JavaScript, riesce comunque a gestire operazioni concorrenti grazie all'event loop e al sistema di I/O non bloccante.

**Eventi e Callback**: Quando un evento si verifica (lettura di un file completata), l'event loop richiama una funzione di callback associata a quell'evento. Questa funzione di callback esegue il codice necessario per gestire l'evento.

**Vantaggi**:

* **Efficienza**: I/O non bloccante.
* **Scalabilità**: gestire molte connessioni simultaneamente senza la necessità di creare nuovi thread per ogni connessione.

**Svantaggi**:

* **Congelamento dell'UI**: Un evento che richiede molto tempo per essere eseguito può congelare l'UI.
* **Timeouts Ritardati**: I timeout potrebbero essere eseguiti in ritardo se ci sono molti eventi da gestire. 