Creative Web Development

Guarise, Degl'Innocenti, Rossi - 2018

Lezione 2

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

Approccio alla programmazione

Teoria

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand”

Scrivere codice **non è solo finalizzato a farlo eseguire dal computer**, ma anche mantenerlo **pulito, coerente, ben organizzato e facile da migliorare e da leggere** per chiunque ci lavorerà sopra ulteriormente.

Bad Code VS Good Code

Bad Code

```
const a = 10
const b = 20
//primo giro
var c = prompt("Inserisci il primo numero da
verificare:")
if( c > a && c < b){
alert("Il numero è nel range.")
}else{
alert("Il numero non è nel range.")
}
//secondo giro
var d = prompt("Inserisci il secondo numero da
verificare:")
if( d > a && d < b){
    alert("Il numero è nel range.")
      }else{
    alert("Il numero non è nel range.")
}
```

Good Code

```
//constants declaration
const minLimit = 10;
const maxLimit = 20;

//function to read a number input from the user
function readNumber(){
    return prompt("Inserisci un numero da
verificare:");
}

//function to check if a number is in the range
function isNumberInRange(number){
    if( number > minLimit && number < maxLimit){
        alert("Il numero è nel range.");
    }else{
        alert("Il numero non è nel range.");
    }
}

//code to effectively run the functions
isNumberInRange(readNumber());
isNumberInRange(readNumber());
```

Coding Best Practice

- Scrivere in **inglese**!
- Utilizzare **commenti ove necessario** per spiegare meglio il codice
- Non commentare dove non necessario
- Utilizzare sempre **l'indentazione** corretta e in modo consistente
- **Raggruppare** le linee di codice in blocchi per ogni suo singolo compito
- Utilizzare una **nomenclatura** consistente per variabili e funzioni
- *DRY*: ***Don't repeat yourself.*** **Riutilizzare** sempre lo stesso codice dove possibile invece di copia-incollarlo (anche dinamicizzandolo)
- **Non annidare troppo** in profondità *(max. 2-3 livelli)*
- **Non scrivere** linee di codice **troppo lunghe**
- Leggere del codice open source per farsi un'idea delle pratiche diffuse

Programmazione procedurale VS programmazione ad oggetti

- **Paradigma di programmazione** = stile fondamentale di programmazione
- Definisce il modo in cui il programmatore concepisce e percepisce il programma stesso
- **Concetti e le astrazioni** usate per rappresentare gli elementi di un programma
- Il modo in cui il programma interagisce nelle sue varie parti
- Scelta del parasigma in base della necessità di **complessità** del programma e della **velocità di scrittura** dello stesso, o del **linguaggio utilizzato**
- Semplice e veloce → Programmazione Procedurale
- Complesso e ordinato → Programmazione ad Oggetti

Programmazione procedurale

- **Blocchi di codice racchiusi** da delimitatori (solitamente parentesi graffe { })
- Ogni blocco ha un suo nome specifico che lo identifica, e viene chiamato “**procedura**”, “**funzione**” o “**sottoprogramma**”
- Ogni funzione, legge in **input**, **esegue** delle operazioni, e restituisce in **output**
- Ogni funzione ha un suo “**scope**”, che isola il suo funzionamento dal resto del codice del programma (**incapsulamento**)
- **Input** = Parametro o variabile
- **Output** = Valore di ritorno
- Linguaggi: C, PHP, JS

Esempio: Programmazione Procedurale

```
var input1 = prompt("Scrivi un numero:");
var input2 = prompt("Scrivi un altro numero:");

var result = parseInt(input1) + parseInt(input2);

alert("Il risultato è: "+ result);
```

Programmazione ad Oggetti

- **OOP**, Object Oriented Programming
- Prevede di raggruppare in un'unica entità (**classe**) le **strutture dati** e le **procedure** che operano su di esse, creando un "oggetto"
- Dotata di **attributi** (le variabili) e **metodi** (le funzioni) che operano sui dati dell'oggetto stesso.
- Ogni oggetto può interagire con altri oggetti a seconda dei casi di **visibilità**
- **Classe** = Modello dell'oggetto da rappresentare
- **Oggetto** = Istanza di una classe valorizzata e pronta all'uso
- Un oggetto può ereditare caratteristiche da un suo genitore (**ereditarietà**) e implementare caratteristiche di un interfaccia (**polimorfismo**)
- Linguaggi: C++, Java, PHP, JS, ...

Esempio: Programmazione ad oggetti

```
var Reader = {
    read: function(){
        return prompt("Scrivi un numero:");
    }
};

var Elaborator = {
    operand1: null,
    operand2: null,
    result: null,
    elaborate: function(){
        this.result = parseInt(this.operand1) +
parseInt(this.operand2);
    }
};

var Displayer = {
    display: function(output){
        alert("Il risultato è: "+ output);
    }
};
```

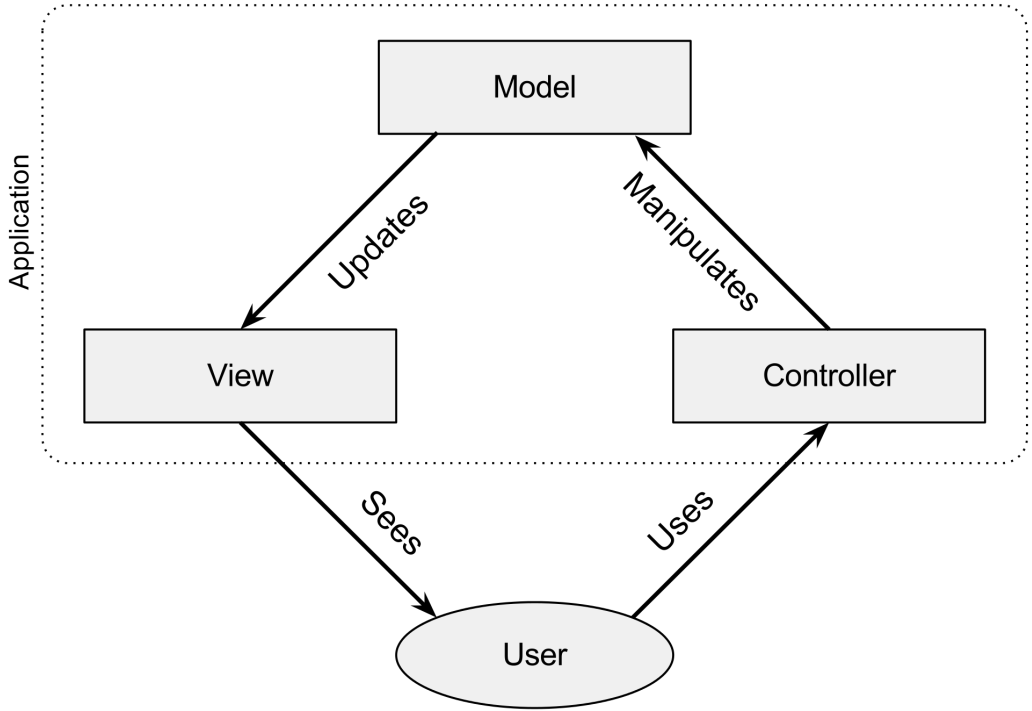
```
var SumController = {
    init: function(){
        var reader = Object.create(Reader);
        var elaborator = Object.create(Elaborator);
        var displayer = Object.create(Displayer);

        //use a temp variable
        var readedValue = reader.read();
        elaborator.operand1 = readedValue;

        //don't use a temp variable but return directly
in-object
        elaborator.operand2 = reader.read();

        elaborator.elaborate();

        displayer.display(elaborator.result);
    }
}

//start code execution
var sumController = Object.create(SumController);
sumController.init();
```

Cos'è l'Architettura Software

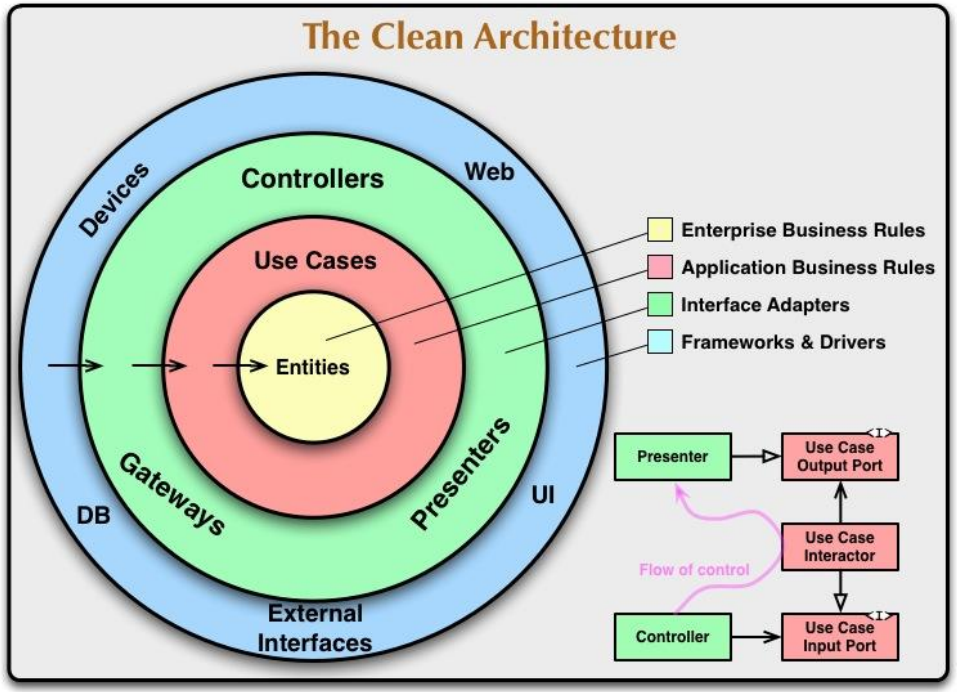
L'architettura software è l'**organizzazione fondamentale di un sistema**, definita dai suoi **componenti**, dalle **relazioni reciproche tra i componenti** e con l'ambiente, e i principi che ne governano la progettazione e **l'evoluzione**.

- Individuazione delle **sottoparti del sistema**
- Individuazione delle **relazioni tra le sottoparti** del sistema
- Strategia condivisa da tutte le parti del sistema per l'organizzazione dello stesso **(Pattern architetturale)** al fine di separare i compiti dei vari elementi

Esempio di Pattern: MVC



Esempio di Pattern: Clean Architecture



La Nomenclatura

Anche nota come **Style Guide**, o **Naming Convention**

Per il JS: https://www.w3schools.com/js/js_conventions.asp

Utile conoscere anche le **Best Practices:**

https://www.w3schools.com/js/js_best_practices.asp

Pratica

Percorso

- Inserire un JS in un documento HTML
- L’istruzione
- Blocchi di codice
    - costrutti condizionali
        - if
        - if else
        - if else if else
        - switch/case
    - costrutti iterativi
        - for
        - foreach
        - while
    - funzioni
    - metodi di oggetti
- Il flow di esecuzione
    - function call
    - return

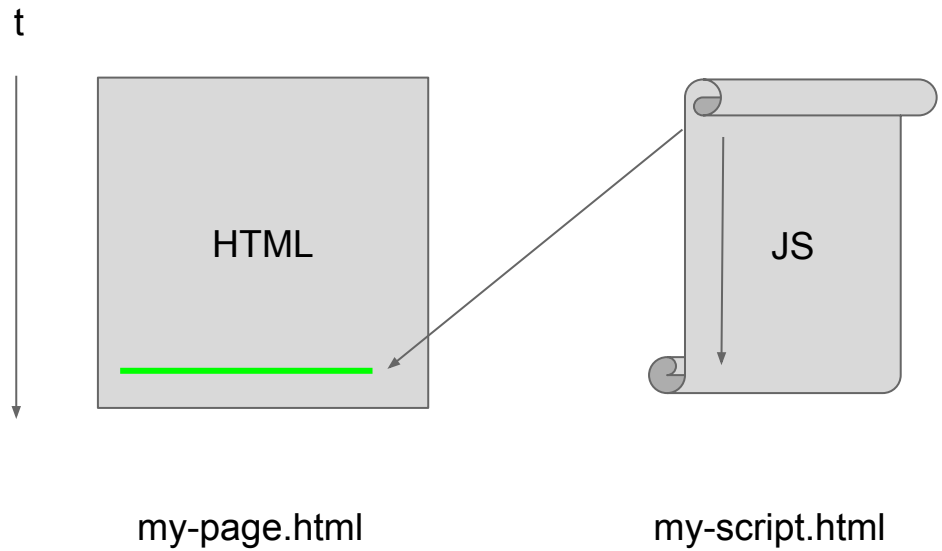
Inserire un JS in un documento HTML

Per inserire uno script javascript all'interno di un documento ipertestuale navigabile su internet, è necessario includere **all'interno di un documento HTML un tag `<script>`** con il riferimento all'URL dello script stesso come di seguito mostrato, **prima della chiusura del tag `body`** del documento.

```
<html>
<head><!-- head code … --></head>
<body>
<!-- body code … -->
<script type="text/javascript" src="my_script.js" ></script>
</body>
</html>
```

Inserire un JS in un documento HTML

All’interno del documento JS è possibile scrivere del codice che verrà subito eseguito durante il caricamento della pagina.



Inserire un JS in un documento HTML

In alternativa è possibile **inserire il codice direttamente in pagina**, seppur non sia consigliato se non in pochissimi casi.

```
<html>
<head><!-- head code … --></head>
<body>
<!-- body code … -->
<script>
alert("I'm executing!");
</script>
</body>
</html>
```

L'istruzione

Unità fondamentale di informazione comprensibile dall'elaboratore, atta a fargli svolgere un determinato compito.

- Ha un punto di ingresso e un punto di uscita
- Può far riferimento al suo interno ad altre istruzioni
- Viene organizzata in blocchi di istruzioni

DOCUMENTAZIONE DI RIFERIMENTO PER JAVASCRIPT:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements

L'istruzione

- **Può eseguire operazioni di molteplici tipi:**
  - Elaborazioni matematiche: `+ - * / %`
  - Dichiarazione e assegnazione di variabili: `var, const, =`
  - Operazioni logiche: `==, ===, !=, !==, &&, ||`
  - Cicli e condizionalità: `if, else, switch/case, for, foreach while`
  - Input ➜ `prompt()`
  - Output ➜ `alert() , console.log(), document.write()`
  - *Tanto altro...* (operazioni di rete, su disco, in memoria, grafiche, elaborazioni di vario tipo...)

L'istruzione in JS

- Una riga di codice dovrebbe essere occupata da una sola istruzione, per semplificare la lettura e l'ottimizzazione del codice
- L'istruzione dovrebbe sempre terminare con un punto e virgola **;**
- Case Sensitive: `variableName` non è lo stesso di `Variablename`

Blocchi di codice

Sottoinsieme di istruzioni adiacenti raggruppate in un blocco.

Viene delimitato dalle parentesi graffe { }

```
{
    istruzione1;
    istruzione...;
    istruzioneN;
}
```

Blocchi di codice: costrutti condizionali

Il costrutto condizionale permette di **modificare il flusso di esecuzione del codice** eseguendo opzionalmente un blocco di codice, o eseguendo un blocco di codice piuttosto che un altro.

Sono costrutti condizionali le seguenti istruzioni:

- `if`
- `if .. else`
- `if .. else if .. else`
- `switch/case`

Costrutti condizionali: if

```
if ( <condizione> ) {
    <blocco di codice>
}
```

Costrutti condizionali: if

Esegue il codice interno al blocco nel caso in cui la **condizione specificata tra parentesi tonde sia VERA**.

```
//other code
if( aNumberVariable > 10 ){
    alert("It's greater than 10");
    callSomeFunction();
    //other code
}
//other code
```

CONDIZIONE

BLOCCO DI CODICE

COSTRUTTO CONDIZIONALE

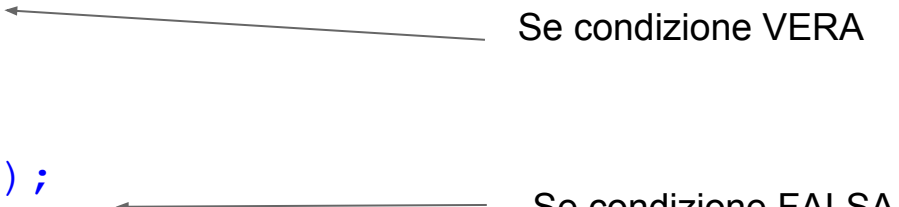
Costrutti condizionali: if … else

```
if ( <condizione> ) {
    <blocco di codice>
}else{
    <blocco di codice>
}
```

Costrutti condizionali: if ... else

Un altra forma di **if**. Permette di specificare un **blocco di codice alternativo** che viene quindi eseguito quando la condizione risulta **FALSA.**

```
if( aNumberVariable < 10 ) {
    doSomething();
    //...
} else {
    doSomethingElse();
    //...
}
```



Costrutti condizionali: if … else if … else

```
if ( <condizione> ) {
    <blocco di codice>
}else if( <altra cond.> ){
    <blocco di codice>
}else{
    <blocco di codice>
}
```

Costrutti condizionali: switch/case

```
switch ( <variabile> ) {
     case <valore>:
          <blocco di codice>
          break;
    case <valore>:
          <blocco di codice>
          break;
    default:
          <blocco di codice>
}
```

Costrutti condizionali: switch/case

Un altro tipo di **costrutto condizionale**. Permette di selezionare un blocco di codice da eseguire in funzione del **valore di una variabile rispetto a dei valori stabiliti**.

```
switch( aVariable ) {
    case 1: doSomething();
        //...
        break;
    case 2: doSomething();
        //...
        break;
    case 7: doSomething();
        //...
        break;
}
```

Se aVariable vale 1

Se aVariable vale 2

Se aVariable vale 7

Costrutti condizionali: switch/case

Il blocco di codice corrispondente inizia dal label del `case` e termina all'istruzione di `break`.

```
switch( aVariable ) {
     case 1: doSomething();
          //…
          break;
     case 7:
          doSomething();
          //…
          break;
}
```

Inizio del blocco in corrispondenza del `case`

Blocco di codice

Fine del blocco in corrispondenza del `break`

Costrutti condizionali: switch/case

Uno stesso blocco di codice o parte di esso può essere **condiviso tra più `case`**, fino al termine dato da un `break`

```
switch( aVariable ) {
    case 1:
        doSomething1();
    case 2:
        doSomething2();
        break;
    case 5:
    case 6:
    case 9:
        doSomething3();
        break;
}
```

E' possibile definire un caso **`default`** che eseguirà tutte le casistiche non altrimenti coperte dai `case`

```
switch( aVariable ) {
    case 1:
        doSomething1();
        break;
    case 7:
        doSomething2();
        break;
    default:
        doSomethingElse();
}
```

Esercizi: Costrutti condizionali

1. Letto un numero, stampare un messaggio in caso il numero sia uguale a 10.
2. Letti due numeri, stampare un messaggio in caso siano uguali tra di loro.
3. Letti due numeri, stampare un messaggio in caso siano uguali tra di loro oppure un altro messaggio in caso siano diversi.
4. Letti due numeri, stampare un messaggio in caso siano uguali tra di loro, un altro messaggio in caso il primo sia maggiore, un altro ancora in caso in cui il secondo sia maggiore.
5. Leggere una parola, se “pizza” o “snack” stampare “pausa veloce”, se invece “bistecca” ➜ “pausa lunga”, altrimenti “pausa generica”.

Blocchi di codice: Costrutti iterativi

Il costrutto iterativo permette di **ripetere l'esecuzione dello stesso blocco di codice più volte**, al fine di ripetere una stessa operazione fino al raggiungimento di un risultato voluto. Spesso ad ogni ciclo del costrutto, si accompagna il cambio di stato di **una variabile che viene adoperata in quel blocco.**

Sono costrutti iterativi i seguenti:

- `for`
- `foreach`
- `while`

Costrutti iterativi: for

```
for( <inizializ.> ; <operaz.> ; <condiz.> ) {
    <blocco di codice>
}
```

Costrutti iterativi: for

Esegue u**n blocco di codice più volte** a seconda di un **espressione**.

- **Inizialmente**, **dichiara e inizializza** una variabile ad un valore di partenza.
- **Poi**, specifica l'**operazione da eseguire sulla variabile** al termine di ogni ciclo.
- **Infine**, specifica una **condizione da soddisfare sulla variabile** per continuare con il ciclo successivo.

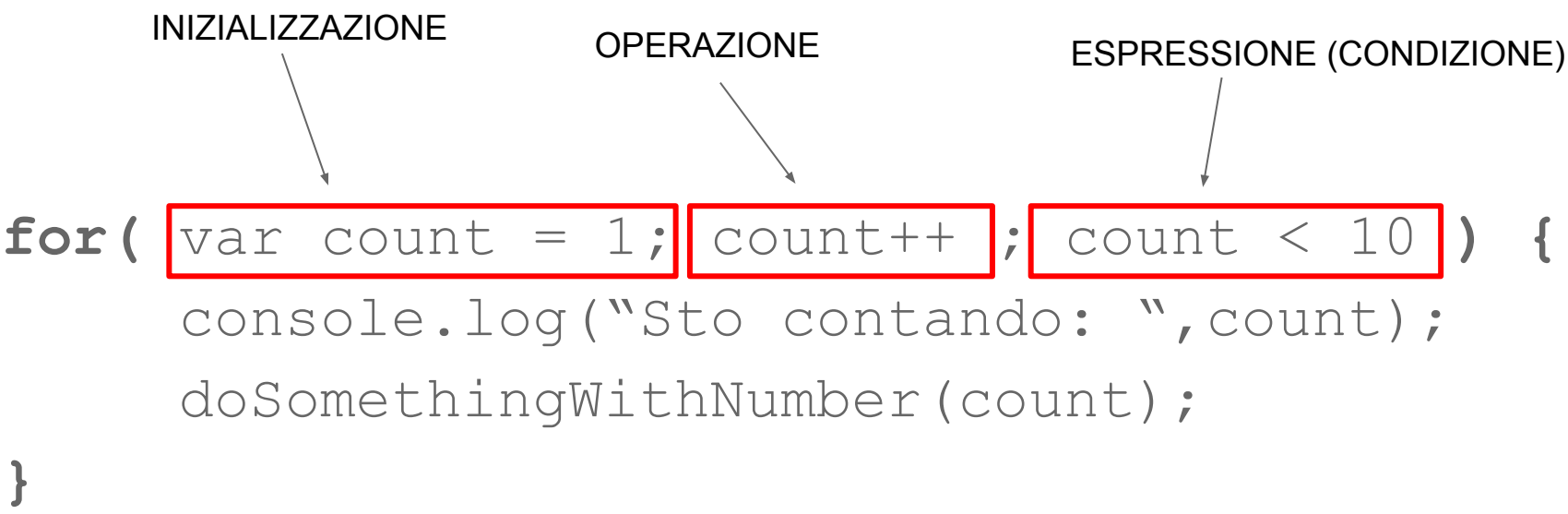
ESPRESSIONI

```
for( var count = 1; count++ ; count < 10) {
    console.log("Sto contando: ",count);
    doSomethingWithNumber(count);
}
```

BLOCCO DI CODICE

COSTRUTTO ITERATIVO

Costrutti iterativi: for



Costrutti iterativi: for

```
for( var count = 1; count++ ; count < 10 ) {
     console.log("Sto contando: ",count);
     doSomethingWithNumber(count);
}
```

Output:

```
Sto contando: 1
Sto contando: 2
Sto contando: 3
Sto contando: 4
Sto contando: 5
Sto contando: 6
Sto contando: 7
Sto contando: 8
Sto contando: 9
```

Costrutti iterativi: foreach

```
var <array> = [<elemento>,<elemento>];

for( var <elemento> in <array> ) {
    <blocco di codice>
}
```

Costrutti iterativi: foreach

Esegue un **blocco di codice più volte** a seconda degli elementi all'interno di vettore o di un di oggetto (**oggetti enumerabili**).
Specifica nell'espressione il nome dell'elemento in corso osservabile nel singolo ciclo.

```
var array = [1,2,3,4];
for( var element in array ) {
    console.log("Elemento: ",element);
    doSomethingWithNumber(count);
}
```

ESPRESSIONE

BLOCCO DI CODICE

COSTRUTTO ITERATIVO

Costrutti iterativi: while

```
while( <condizione> ) {
    <blocco di codice>
}
```

Costrutti iterativi: while

Esegue un **blocco di codice più volte** fintanto che la condizione risulta VERA.

```
var number = 0;
while( number < 10 ) {
      console.log("Elemento: ",i);
      doSomethingWithNumber(count);
      number++;
}
```

CONDIZIONE

BLOCCO DI CODICE

COSTRUTTO ITERATIVO

Esercizi: Costrutti iterativi

1. Utilizzare il ciclo FOR per stampare i numeri da 15 a 30 inclusi.
2. Utilizzare il ciclo FOR per stampare i numeri da 100 a 90 inclusi.
3. Utilizzare il ciclo FOREACH per stampare tutti i valori dell'array [1,3,5,7,9,11,13,15].
4. Utilizzare il ciclo WHILE per stampare i numeri da 42 a 46.

Blocchi di codice: le funzioni

- Detta anche procedura, sottoprogramma, metodo, routine o subroutine
- Racchiude più istruzioni al suo interno
- Serve ad operare una **specifica operazione** od elaborazione
- Può ricevere uno o più dati in **input** da elaborare
- Può restituire un dato in **output**
- Può essere **richiamata più volte da qualsiasi punto del codice** come fosse una singola istruzione ➜ **Riuso del codice**
- Al suo interno è possibile dichiarare **variabili non accessibili dall'esterno** del codice della funzione stessa
- La comunicazione di valori avviene attraverso **parametri di ingresso** e **valori di ritorno**

Blocchi di codice: le funzioni

```
function <function name>( <parameters> ){
    <code block>
    <code block>
    return <value or variable>
}
```

Blocchi di codice: le funzioni

```
function doSomething( aParameter ){
    console.log(“Parametro in ingresso: ”,aParameter);
    return aParameter+1;
}
```

Blocchi di codice: le funzioni

```
function doSomething( aParameter, anotherParameter ){
    console.log(“Parametro in ingresso: ”, aParameter);
    console.log(“Altro param. in ingresso: ”, anotherParameter);
    return;
}
```

Blocchi di codice: le funzioni

PARAMETRI IN INGRESSO

```
function doSomething( aParameter, anotherParameter ){
    console.log("Parametro in ingresso: ", aParameter);
    console.log("Altro param. in ingresso: ", anotherParameter);
    result = aParameter + anotherParameter;
    return result;
}
```

VALORE DI RITORNO

Blocchi di codice: le funzioni

```
function doSomething( aParameter, anotherParameter ){
    console.log("Parametro in ingresso: ", aParameter);
    console.log("Altro param. in ingr.: ", anotherParameter);
    return aParameter + anotherParameter;
}

//PER RICHIAMARE (UTILIZZARE) LA FUNZIONE
doSomething( 10, 15 );
//OPPURE
var result = doSomething( 10, 15 );
```

Blocchi di codice: le funzioni

```
function multiplication( factor1, factor2 ){
    return factor1 * factor2;
}

var read1 = prompt("Primo valore?");
var read2 = prompt("Secondo valore?");
var result = multiplication(read1,read2);
console.log( "Risultato: ", result );
```

Esercizi: Le funzioni

1. Creare e richiamare una funzione che quando richiamata stampi “Hello World”
2. Creare e richiamare una funzione che dato un numero in ingresso lo stampi in console
3. Creare e richiamare una funzione che dati due numeri in ingresso ne stampi la somma in console
4. Creare una funzione che dati due numeri in ingresso ne restituisca la somma, e nel corpo del programma (fuori dalla funzione) stampare il risultato.

Blocchi di codice: metodi di oggetti

- Un metodo è un tipo di **funzione tipica di un oggetto**.
- Ha lo stesso comportamento di una funzione, ma il suo utilizzo deve venir considerato all'interno della metodologia di **programmazione ad oggetti.**
- Vedremo in seguito il suo utilizzo dettagliato.

Blocchi di codice: metodi di oggetti

// esempio di metodo in un oggetto che prende in input un valore e lo restituisce sommato di 1

```
var AnObject = {
    attribute: 12,
    method: function( aParameter ){
        return aParameter+1;
    }
};

var instance = Object.create(AnObject);
var result = instance.method(12);
```

Blocchi di codice: Il flow di esecuzione

- Per flow di esecuzione di un **blocco di codice** si intende:
  - **INGRESSO:** Dove il blocco inizia la propria esecuzione (con eventuali parametri in ingresso) ***(function call)***
  - **USCITA:** Dove il blocco conclude la propria esecuzione (con eventuali valori di ritorno) e ritorna al codice dove è stato chiamato in origine ***(return)***

Blocchi di codice: Il flow di esecuzione

```
function multiplication( factor1, factor2 ){
    return factor1 * factor2;
}
```

USCITA

```
var read1 = prompt("Primo valore?");
var read2 = prompt("Secondo valore?");
var result = multiplication(read1,read2);
```

INGRESSO

Esercizi: Flow di esecuzione

1. Realizzare il seguente script e avviarlo in debug, seguendone il flusso di esec.

```
function printAndSum( addend1, addend2 ){
    console.log("Addendo:",addend1);
    console.log("Addendo:",addend2);
    var sum = parseInt(addend1) + parseInt(addend2);
    return sum;
}

var read1 = prompt("Primo numero: ");
var read2 = prompt("Secondo numero: ");
debugger;
var result = printAndSum(read1,read2);
console.log(result);
```

Esercizi da svolgere a casa

1. *IF/ELSE* : Realizzare uno script che letto un numero da tastiera, stampi se il numero è maggiore o minore di 15.
2. *SWITCH/CASE* : Realizzare uno script che letto un numero da tastiera, se 15 o 16 stampi un messaggio “numero trovato”, altrimenti “numero non trovato”.
3. *FOR*: Realizzare uno script che letto un numero da tastiera stampi tutti i numeri da 1 fino a quel numero
4. *FUNCTION*: Realizzare uno script che letti due numeri, li passi ad una funzione come parametri di ingresso, la quale li sommerà e restituira il risultato; stampare poi il risultato dal codice del corpo del programma (non dalla funzione).