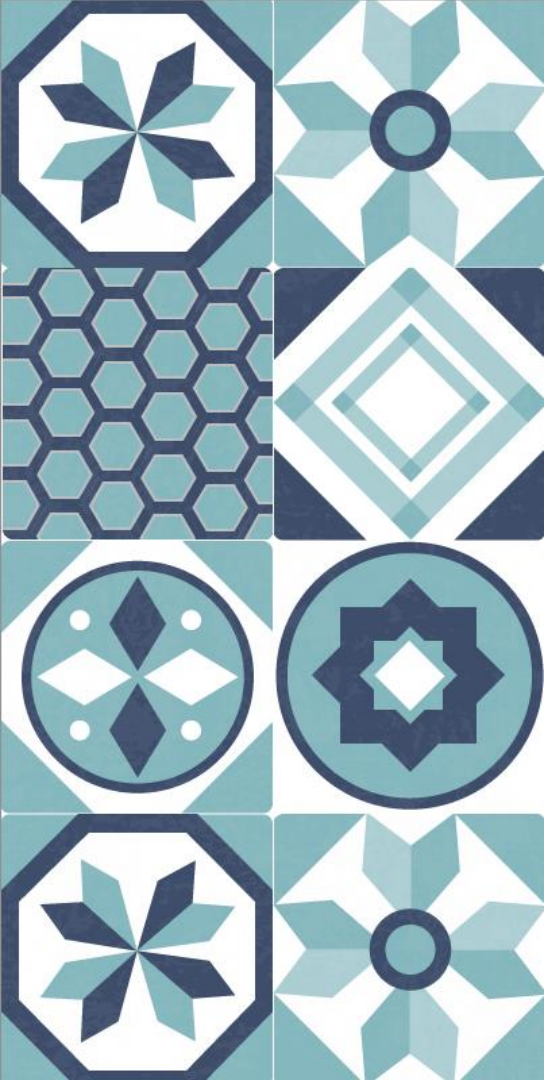
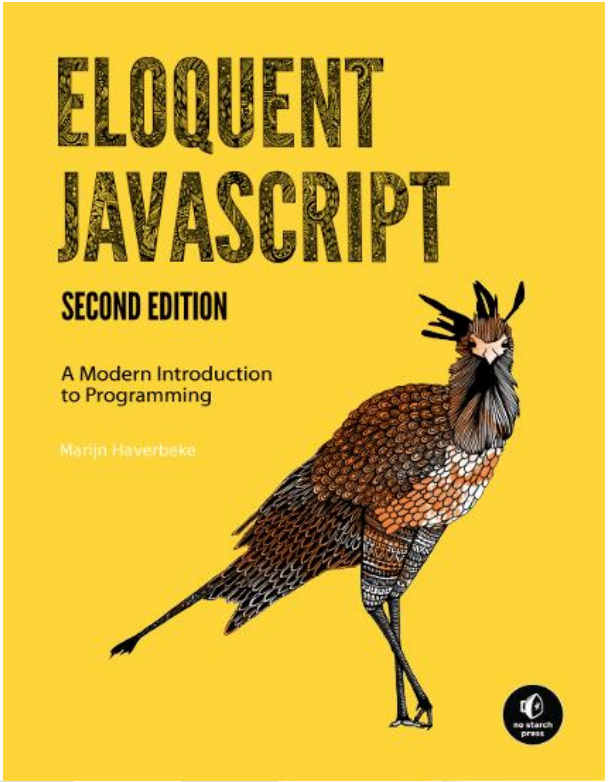


FONDAMENTI DI INFORMATICA

Alma Artis
Francesca Pratesi (ISTI, CNR)

Introduzione a Javascript

LIBRI E RIFERIMENTI



- Capitolo 1
- Capitolo 2

Eloquent Javascript – Second Edition
Marijn Haverbeke

Available here: http://eloquentjavascript.net/

INTRODUZIONE A JAVASCRIPT

PUNTI DI FORZA

- JavaScript è il linguaggio di programmazione più popolare al mondo
- JavaScript è il linguaggio di programmazione del Web (linguaggio di scripting, permette di rendere dinamiche ed interattive le pagine web)
- JavaScript è facile da imparare

ALTRE CARATTERISTICHE

- Una volta imparato a programmare in qualsiasi linguaggio (anche in Javascript) è facile imparare anche altri linguaggi di programmazione
  - Anche se alcune possibilità sono prerogrative solo di alcuni lunguaggi o famiglie di linguaggi
- Consente di esprimere tutte le funzioni calcolabili
- L'interprete è incorporato nei web browser (e servers)
- Weakly typed (tipizzazione debole)
- Javascript ≠ Java
- Javascript è un linguaggio interpretato
  - Serve un interprete Javascript per eseguire il codice
  - Es: https://replit.com/

ALTRE CARATTERISTICHE (2)

- Un programma in JavaScript è una sequenza di comandi
- *JavaScript è un linguaggio imperativo*
  - I comandi (o istruzioni) sono eseguiti uno dopo l'altro nell'ordine in cui compaiono
- I comandi semplici devono terminare con un punto e virgola (;)
- JavaScript è *case sensitive*: fa distinzione tra lettere maiuscole e minuscole
  - *log e Log sono diversi!*

COMMENTI

- Possibilita’ 1:
  - Il testo racchiuso tra /* e */ è un commento, ad esempio
  - /*Stampa Ciao mondo! in output */
- Possibilita’ 2:
  - Il testo che segue // fino alla fine della riga è un commento, ad esempio
  - //Il mio primo programma
- L’interprete ignora i commenti
- I commenti sono fondamentali per rendere comprensibile il programma a chi lo scrive e a chi lo legge

ISTRUZIONE DI STAMPA (OUTPUT)

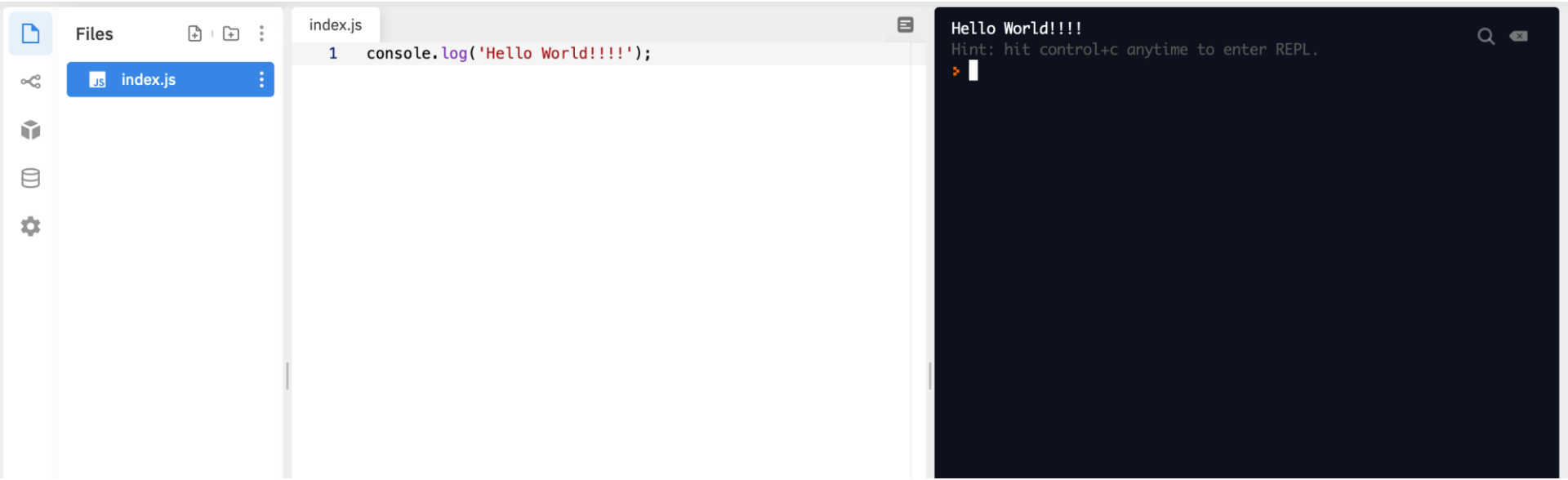
- console.log("Ciao mondo!");
- Stampa l'argomento della funzione in output (solitamente nella console del browser)
- In questo caso l'istruzione chiede di visualizzare sulla finestra di output la sequenza di caratteri tra gli apici

KEYWORDS E PAROLE RISERVATE

- Parole con un significato particolare, come le keywords, non possono essere utilizzate come nomi di variabili
- Altre parole sono riservate per uso futuro del linguaggio

```
break case catch class const continue debugger
default delete do else enum export extends false
finally for function if implements import in
instanceof interface let new null package private
protected public return static super switch this
throw true try typeof var void while with yield
```

IL NOSTRO PRIMO PROGRAMMA



ESPRESSIONI

- Una espressione è un frammento di codice che produce un valore
- Ogni valore di base è una espressione (es. 22, “programma”,3.14)
- Una espressione tra parentesi è ancora una espressione
- Un operatore binario applicato a due espressioni produce una espressione
- Un operatore unario applicato ad una espressione produce una espressione
- Espressioni più semplici possono essere combinate per creare una espressione più complessa

ISTRUZIONI (O STATEMENT)

- L'esempio più semplice di istruzione è una espressione seguita da ";"
  - 1;
  - !false;
- Una istruzione può essere fine a se stessa (vedi i due esempi qui sopra)
- Può ritornare un risultato sullo schermo (vedi istruzione console.log)
- Può cambiare lo stato interno del calcolatore in modo da influenzare le istruzioni che verranno dopo
- In molti casi è essenziale terminare una istruzione con il simbolo ";"
- Ci sono casi in cui non è necessaria. Per evitare problemi, tendiamo a inserirlo sempre

VALORI, TIPI E OPERATORI

RAPPRESENTAZIONE DELL'INFORMAZIONE

- Il calcolatore può leggere, modificare e gestire solo dati
- I dati sono rappresentati come sequenze di bit, ovvero sequenze di 1 e 0
- I valori 1 e 0 sono rappresentati dal calcolatore come tensioni elettriche, rispettivamente alta e bassa
- Rappresentazione di numeri basata su base 2 invece che su base 10
- Ad esempio, il numero 13:
  - 0 0 0 0 1 1 0 1
  - 128 64 32 16 8 4 2 1

ESPRESSIONI

- Un’espressione in JavaScript rappresenta un calcolo che restituisce un valore
  - Una costante (o letterale) è un’espressione semplice
  - Un’espressione composta si ottiene combinando una o più espressioni tramite un operatore
  - Un operatore combina uno, due o più operandi

VALORI E TIPI

- In un calcolatore ci sono sequenze lunghissime di bit
- Sequenze di bit sono organizzate in segmenti logici, chiamati valori
- Diversi valori possono avere diversi ruoli
  - Numeri, stringhe, boolean, oggetti, funzioni, e undefined
- Ogni valore viene creato nel momento in cui viene invocato
- Ogni valore viene conservato nella memoria interna del calcolatore, fino a quando c'è spazio disponibile
- I valori che non vengono utilizzati vengono distrutti e il loro spazio può essere riutilizzato

COSTANTI NUMERICHE

- I numeri sono rappresentati da sequenze di cifre
  - Ad esempio il numero 13
- Nel momento in cui viene incontrato il numero 13 in un programma, viene creato uno spazio in memoria in cui sono conservati i bit che lo rappresentano
- Ogni numero in Java viene rappresentato con 64bit
  - Possiamo rappresentare 2^{64} valori differenti per singolo valore
  - Dobbiamo rappresentare anche il segno (negativo o positivo) e la posizione della cifra decimale
  - Operazioni in virgola mobile perdono precisione quando rappresentati su uno spazio finito

COSTANTI NUMERICHE (2)

- Le costanti numeriche sono:
  - interi: es. 456, 0, -7, +42
  - razionali (numeri con una parte decimale non nulla)
- I numeri razionali si possono rappresentare
  - con notazione decimale, es: 3.14, -4.12, 0.271
  - con notazione esponenziale (o scientifica), es:
    - 3E+2 = 300, 314E-2 = 3.14
    - 2.998e8 = 2.998 * 10^8= 299,800,000
- Numeri speciali:
  - Infinity e –Infinity rappresentano rispettivamente infinito positivo e negativo
- Conversione automatica da interi a razionali quando necessario
  - es: 1 viene automaticamente convertito in 1.0, ecc

OPERATORI NUMERICI (O ARITMETICI)

- Operatori unari (prendono in input un numero e producono in output un numero)
  - + si applica a costanti numeriche e non ha effetto, es.
    - +4, +3.14, +34
  - - si applica a costanti numeriche e ne cambia il segno,
    - es. -3.14, -4, -34

OPERATORI NUMERICI (2)

- Operatori binari (prendono in input due numeri e producono in output un numero)
  - *+ addizione, es. 3 + 4 = 7*
  - *- sottrazione, es. 10 – 7 = 3*
  - ** moltiplicazione, es. 30 * 2 = 60*
  - */ divisione, es. 3/2 = 1.5*
  - *% modulo (resto della divisione intera)*
    - *3 % 2 = 1*
    - *3 % 3 = 0*
  - *** elevamento a potenza, es. 2 ** 3 = 8*

OPERATORI NUMERICI (3)

- Gli operatori possono esser concatenati
  - `100 + 4 * 11`
- Gli operatori hanno una precedenza. In questo caso l'operatore di moltiplicazione viene eseguito prima dell'operatore moltiplicazione
- Moltiplicazioni e divisioni hanno la precedenza su addizione e sottrazione
- La lista di operatori precedenti è in ordine di precedenza decrescente
- Si può cambiare l'ordine di precedenza utilizzando le parentesi
  - `(100 + 4) * 11`
- È possibile «incapsulare» più parentesi, ma sempre tonde (quadre e graffe hanno un altro significato)
  - `(100 + (2+4)) * 11`

COSTANTI (LETTERALI) STRINGHE

- Una stringa è una sequenza di caratteri delimitata da un apice doppio (”) o singolo (‘), ad esempio:
  - *"Lorem ipsum dolor sit amet"*
  - *'Lorem ipsum dolor sit amet’*
- I caratteri sono tutti i *caratteri stampabili*:
  - *le lettere alfabetiche minuscole e maiuscole, le cifre numeriche (da non confondere con i numeri), i segni di interpunzione e gli altri simboli che si trovano sulla tastiera di un calcolatore (e qualcuno di più)*
- Le costanti stringhe devono essere scritte su una sola riga

SEQUENZE DI ESCAPE

- A volte può essere necessario stampare dei caratteri/comandi speciali; ad esempio: ‘, “,
- Sequenze di escape: sequenza di 2 caratteri in cui il primo carattere è \
  - \’ *stampa* ‘
  - \" *stampa* "
  - \\ *stampa* \
  - \n *indica ritorno a capo*
  - \t *indica tabulazione orizzontale*
- `console.log(‘Egli disse \’ciao\’ appena la vide.’)`

CONCATENAZIONE DI STRINGHE

- Per concatenare due stringhe si usa l'operatore +
- Operatore binario (si applica a due operandi)
- Esempi:
  - "Java" + "Script" = "JavaScript"
  - "2" + "3" = "23"
  - console.log("con"+"cat"+"e"+"nate");
    // → "concatenate"

COSTANTI LOGICHE O BOOLEANE

- Sono usate per rappresentare valori di verità
  - VERO → true / 1
  - FALSO → false / 0
- Non sono stringhe!!!!!

OPERATORI BOOLEANI

- Operatori unari
  - ! negazione (o complemento), cambia il valore di verità
- Operatori binari
  - && congiunzione, true solo se entrambi gli operandi sono true
  - || disgiunzione, true se almeno uno degli operandi è true
- La valutazione degli operatori è lazy (pigra): il secondo operando viene valutato solamente se è necessario
  - *per &&: solo se il primo operando è true,*
  - *per ||: solo se il primo operando è false.*

OPERATORI BOOLEANI (2)

- L'operatore && restituisce `true` se entrambi i suoi argomenti sono `true`

```
console.log(true && false)
// → false
console.log(true && true)
// → true
console.log(false && false)
// → false
console.log(false && true)
// → false
```

- L'operatore || restituisce `true` se uno dei suoi argomenti è `true`

```
console.log(false || true)
// → true
console.log(false || false)
// → false
console.log(true || true)
// → true
console.log(true|| false)
// → true
```

OPERATORI BOOLEANI (3)

- L'operatore ! restituisce il valore inverso del suo argomento

```
console.log(!false)
// → true
console.log(!true)
// → false
```

- L'operatore condizionale (cond? val1 : val2) è un operatore ternario. Se il valore a sinistra del simbolo ? è vero ritorna il primo valore, altrimenti il secondo

```
console.log(true ? 1 : 2);
// → 1
console.log(false ? 1 : 2);
// → 2
```

PRECEDENZA DEGLI OPERATORI LOGICI

- Gli operatori logici && e || valutano o convertono in tipo booleano l'operando a sinistra per determinare il risultato del confronto
- In base al risultato di questa prima valutazione, il comportamento può variare
- Questa proprietà può essere utile per assegnare un valore di default in caso uno sia mancante

```
console.log(null || "user")
// → user
console.log("Karl" || "user")
// → Karl
```

- Come nel caso degli operatori aritmetici, anche per gli operatori logici sono previste delle precedenze
  - il NOT (!) ha la precedenza sull'AND (&&)
  - l'AND (&&) ha la precedenza sull'OR (||)
- Anche in questo caso si può cambiare l'ordine in cui vengono eseguite le operazioni attraverso le parentesi

OPERATORI DI CONFRONTO (O RELAZIONALI)

- Permettono di confrontare 2 valori di un dominio sul quale è definito un ordinamento
  - == uguaglianza
  - != disuguaglianza
  - > maggiore
  - >= maggiore o uguale
  - < minore
  - <= minore o uguale
- Gli operandi devono essere dello stesso tipo

OPERATORI DI CONFRONTO (2)

- Per i numeri l'ordinamento è quello usuale
  - – 2 < 3, 3>=3, 2.2 > 2.0, 2.12 != 4.23435
- Per i booleani
  - false < true
- Per le stringhe si segue l'ordine lessicografico basato sull'ordinamento alfanumerico
  - "abate" < "abete"
  - "solo" > "sola"
  - In realtà è più esatto dire che il confronto tra stringhe segue l'ordinamento dato dalla codifica Unicode, che assegna un valore numerico ad ogni carattere

CONVERSIONI IMPLICITE

- Se un operatore viene eseguito con operandi che non sono del tipo atteso avviene una conversione, se possibile
- Per gli operatori numerici:
  - i Booleani sono convertiti (→) in questo modo
    - true →1
    - false →0
  - le stringhe sono convertite in numeri, se possibile
    - "23" →23, es. 2 * "23" = 46
  - se la stringa non rappresenta un numero viene restituito NaN (not a number), e.g. 2 * '20a' = NaN
- Attenzione all'operatore di concatenazione tra stringhe
- 0 viene convertito in false ("0" non viene convertito!)

CONVERSIONI IMPLICITE (2)

- Se gli operandi non sono booleani:
  - *per &&: se il primo operando non può essere convertito in false allora il risultato è il secondo operando, altrimenti il risultato è il primo operando*
  - *per ||: se il primo operando non può essere convertito in false allora il risultato è il primo operando, altrimenti il risultato è il secondo operando*
- Cosa calcolano le seguenti istruzioni?
  - console.log(5 && true);
  - console.log(5 && false);
  - console.log(0 && true);
  - console.log(0 && false);
  - console.log("0" && true);
  - console.log(5 || true);
  - console.log(5 || false);
  - console.log(0 || false);
  - console.log(0 || true);
  - console.log("0" || true);
  - console.log("0" || false);

VALORI INDEFINITI

- Ci sono due valori speciali che vengono usati per denotare l'assenza di valori significativi
  - `null`
  - `undefined`

SOMMARIO

- Linguaggio Javascript
- Definizione di valori e tipi
- Operatori aritmetici: +, -, *, /, %
- Concatenazione di stringhe: +
- Confronto: ==, !=, !==, <, >, <=, >=
- Operatori logici: &&, || , !
- Operatori unari: -, !
- Operatori ternari: (? :)

ESERCIZI

- Scrivere un programma che stampi un numero, una stringa ed un booleano.
- Scrivere un programma che stampi il cubo del numero 244.
- Scrivere un programma che stampi le potenze di 2 partendo dall'esponente 0 fino all'esponente 5. Il programma deve quindi stampare il risultato delle seguenti espressioni 2^0, 2^1, 2^2, 2^3, 2^4, 2^5.
- Scrivere un programma che calcoli e stampi se il proprio numero di telefono è divisibile per 2.
- Scrivere un programma che calcoli e stampi il lato di un quadrato con perimetro 70 cm.
- Scrivere un programma che calcoli e stampi l'area di un cerchio con raggio 10.5 m.

SCRIVERE UN PROGRAMMA CHE STAMPI UN NUMERO, UNA STRINGA ED UN BOOLEANO.

- ```
  // Questi sono tutti numeri
  console.log(1);
  console.log(100);
  console.log(5.6);
  console.log(-30.5);
  console.log(1+30);
  ```

- ```
  // Queste sono tutte stringhe
  console.log("100");
  console.log("cento");
  console.log("ciao mondo!");
  console.log("ciao"+" "+"mondo");
  console.log("true");
  ```

- ```
  // Questi sono tutti booleani
  console.log(true); // true
  console.log(true&&false); // false
  console.log(true && false); // false
  console.log(3>5); // false
  console.log(true && !false); // true
  ```
  ```

SCRIVERE UN PROGRAMMA CHE CALCOLI E STAMPI SE IL PROPRIO NUMERO DI TELEFONO È DIVISIBILE PER 2

- ```
  // soluzioni parziale
  console.log(3471231234%2); // stampa 0
  console.log(3471231233%2); // stampa 1
  ```

- ```
  // soluzione corretta: controllo che il resto sia uguale a 0
  console.log(3471231234%2==0); // stampa true
  console.log(3471231237%2==0); // stampa false
  ```

- ```
  // se volessi dire se è divisibile per 5?
  console.log(10%5==0); // true
  console.log(11%5==0); // false
  console.log(12%5==0); // false
  console.log(13%5==0); // false
  console.log(14%5==0); // false
  console.log(15%5==0); // true
  ```
  ```

VARIABILI E ASSEGNAMENTO

VARIABILI

- Per mantenere lo stato interno si usano le variabili
- Le variabili permettono di gestire lo stato interno e di memorizzare valori
- Le variabili sono create attraverso la keyword (parola chiave) `var`

```
var caught = 5 * 5;
```

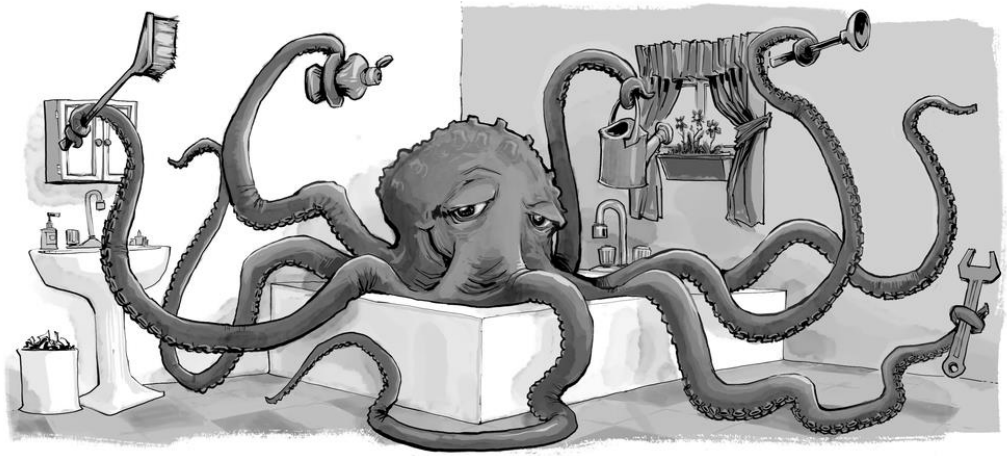
- L'istruzione precedente crea una variabile chiamata caught e la usa per contenere il risultato della moltiplicazione di 5 per 5
- Una volta che una variabile è stata definita, il suo nome può essere utilizzato all'interno delle espressioni successive

```
var ten = 10;
console.log(ten * ten);
// → 100
```

NOMI DI VARIABILI

- Il nome di una variabile può essere qualunque parola
  - non può essere una parola chiave (es `var`)
  - non può contenere spazi
  - Si possono utilizzare numeri, ma il nome non può iniziare con un numero
  - Non si può utilizzare simboli di punteggiatura, ad eccezione di $ e _
- Una variabile non è legata per sempre ad un singolo valore
- L'operatore = può disconnettere una variabile dal suo valore attuale e assegnarne un altro

```
var mood = "light";
console.log(mood);
// → light
mood = "dark";
console.log(mood);
// → dark
```



DICHIARAZIONE E INIZIALIZZAZIONE

- Dichiarazione : la variabile viene creata
  - es.: `var numero;`
- Dopo la dichiarazione il suo valore è «undefined»
- Inizializzazione : la variabile viene dichiarata e le viene assegnato un valore iniziale
  - es.: `var numero = 42;`

DEFINIZIONE MULTIPLA DI VARIABILI

- Una singola parola chiave var può essere utilizzata per definire più variabili
- Ogni definizione deve essere separata da virgola

```
var one = 1, two = 2;
console.log(one + two);
// → 3
```

ASSEGNAMENTO

- Il valore di una variabile può essere cambiato da un assegnamento

```
var numero = 12;
numero = 42;
```

- Dopo la seconda istruzione il valore 12 non è più associato alla variabile numero; di fatto è stato sovrascritto dal valore 42

ESEMPIO

```
var base = 10;
var altezza;
var A;
altezza= 2;
A = base *altezza; //area di un rettangolo
```

IL CONCETTO DI STATO

- Lo **stato** è una rappresentazione dei dati del problema in un dato momento
- La specifica di un problema consiste nella descrizione di uno stato iniziale (che descrive i dati del problema) e di uno stato finale(che descrive i risultati attesi).
- È necessaria una rappresentazione degli oggetti coinvolti (dello stato) manipolabile dall'esecutore prescelto.

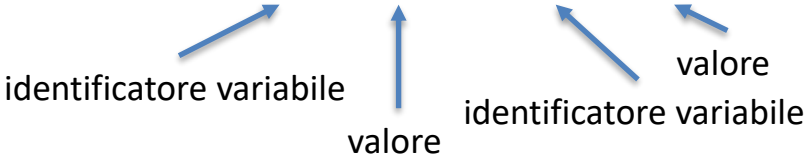
ANCORA SUGLI ALGORITMI

- Un algoritmo è una sequenza di passi elementari che, se intrapresi da un esecutore, comportano ripetute modifiche dello stato fino al raggiungimento dello stato finale desiderato
- Le azioni di base che l'esecutore è in grado di effettuare devono prevedere, tra le altre, azioni che hanno come effetto cambiamenti dello stato

LO STATO

- Lo stato del calcolatore può essere visto come un insieme di associazioni tra identificatori (nomi simbolici) e valori.
- Rappresentiamo lo stato come un insieme di coppie **(*x*, *val*)** dove *x* è l'identificatore e *val* è il valore
- Ad ogni identificatore è associato un solo valore
- Es.

{(numero, 42), (stringa, "ciao!")}



ESEMPI DI STATO

- *{(base, 25), (altezza, 12), (perimetro, 74), (area, 300)}*
- *{(nome,"Marco"), (cognome, "Bianchi"), (eta, 19)}*
- *{(a, 23),(b, 44), (c, undefined)}*
- {(a, 23), (b, 44), (a, 12)}

ESEMPI DI STATO (2)

```
var x,y;
{(x,undefined),(y,undefined)}
x = 0; //assegna 0 alla variabile x
{(x,0),(y,undefined)}
y = x + 1; //calcola il risultato di  x+1  e lo assegna alla variabile y
{(x,0),(y,1)}
y = x; //assegna alla variabile y il valore della variabile x
{(x,0),(y,0)}
```

- Nota: una variabile all'interno di un'espressione sta per il valore ad essa associata in quel momento (cioè: nello stato corrente)

PROGRAMMA COME SEQUENZA DI ISTRUZIONI

- Ogni programma è una sequenza di istruzioni
- Ogni istruzione è eseguita successivamente alla precedente e fa riferimento allo stato corrente del calcolatore
- Ogni istruzione può determinare una modifica nello stato, che dipende dalla semantica dell'istruzione e dallo stato corrente

LA LIBRERIA MATH

LIBRERIE

- Una libreria è una collezione (un insieme) di funzioni già definite e pronte all'uso
- In JavaScript, Math è una libreria di funzioni matematiche

LIBRERIA MATH

- Si può accedere alle funzionalita' di Math indicando come prefisso il nome della libreria seguita dal punto, ad esempio:
  - `Math.pow(2,3);`

- Vediamo adesso alcune funzioni utili

ALCUNI METODI DI MATH

- Math.random();
  - restituisce un numero casuale tra 0 ed 1
  - es: `console.log(random());`
- Math.abs(x);
  - restituisce il valore assoluto di x
  - es. `Math.abs(-1.23);` // restituisce 1.23
- Math.sqrt(x);
  - restituisce la radice quadrata di x
  - es. `Math.sqrt(4);` // restituisce 2
- Math.PI
  - restituisce il valore di PI greco

ALCUNI METODI DI MATH (2)

- Math.round(x);
  - restituisce il valore dell'intero più vicino ad x
  - es. `Math.round(2.6); // restituisce 3`
- Math.floor(x);
  - restituisce il valore dell'intero più vicino ad x, arrotondando per difetto
  - es. `Math.floor(2.9); // restituisce 2`
- Math.ceil(x);
  - restituisce il valore dell'intero più vicino ad x, arrotondando per eccesso
  - es. `Math.floor(2.1); // restituisce 3`

ISTRUZIONI CONDIZIONALI

ISTRUZIONI CONDIZIONALI

- Tutti i programmi in JavaScript visti finora prevedono istruzioni eseguite in sequenza
- Con questi programmi è possibile solo affrontare problemi che richiedano l'esecuzione di una sequenza di calcoli
- A volte è necessario eseguire del codice solo se si verifica una certa condizione
  - Es. Se la temperatura ambientale scende sotto i 18 gradi accendo il riscaldamento

ISTRUZIONI CONDIZIONALI (2)

- Per poter eseguire una istruzione (o un blocco di istruzioni) solo in determinati casi è necessario introdurre le istruzioni condizionali
- Istruzioni condizionali: una istruzione viene eseguita solo se una specifica condizione è verificata
- Condizione: espressione booleana

ISTRUZIONE IF-THEN-ELSE

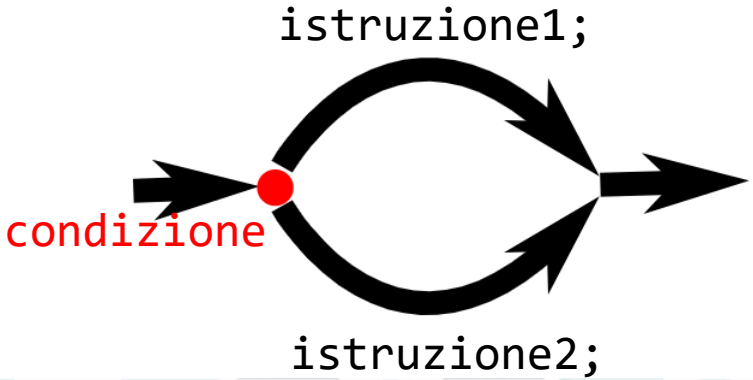
- Sintassi:

```
if (condizione)
    istruzione1;
else
    istruzione2;
```

Espressione booleana

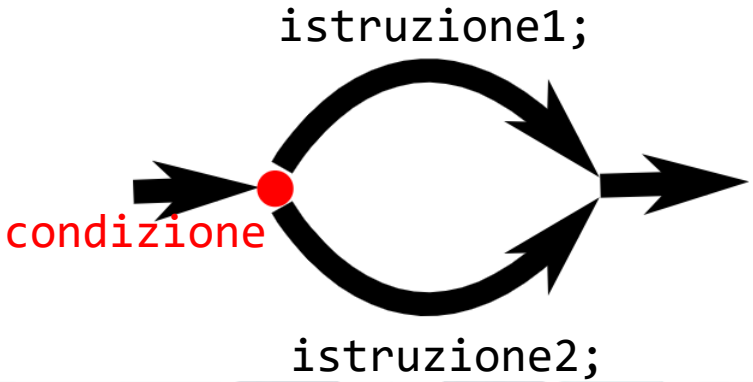
Ramo «then»

Ramo «else»



ISTRUZIONE IF-THEN-ELSE (2)

- Semantica:
  - viene valutata l'espressione booleana «condizione»
  - se «`condizione`» è vera allora viene eseguita l'«`istruzione1`», altrimenti («`condizione`» è falsa) viene eseguita l'«`istruzione2`»
  - successivamente, l'esecuzione riprende dall'istruzione successiva al blocco if-then-else



ESEMPIO

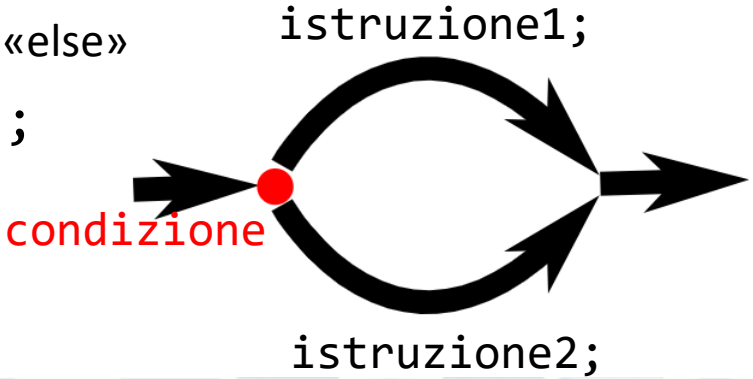
- Scrivere un programma che stampi la stringa "fa caldo" se la temperatura è superiore a 30 gradi e "si sta bene" altrimenti
-

```
var t=27
if (t>30)
    console.log('Fa caldo');
else
    console.log('Si sta bene');
```

Espressione booleana

Ramo «then»

Ramo «else»



ISTRUZIONE IF SEMPLICE (O IF-THEN)

- In alcuni casi (quando non bisogna fare nulla se la condizione è false), non è necessario prevedere un ramo else

- Sintassi:

```
if (condizione)
    istruzione1;
```

Espressione booleana

Ramo «then»

COSA PUÒ CONTENERE LA CONDIZIONE?

- Come detto prima, la condizione è una qualsiasi espressione booleana

COSA PUÒ CONTENERE L'ISTRUZIONE DEI RAMI THEN ED ELSE?

- Una qualsiasi istruzione:
  - sia tra quelle viste finora (es. una stampa, un assegnamento)
  - sia tra quelle che vedremo nelle prossime lezioni
- Nota: l'istruzione può essere anche un if-then o un if-then-else!
  - In questo caso si parla di if annidati
- Ogni ramo può essere composto anche da più istruzioni, che vengono eseguite normalmente
  - In questo caso è necessario inserire il blocco di codice tra parentesi graffe

ESEMPIO

- Scrivere un programma che dati un mese (espresso in numeri da 1 a 12) e un anno stampi il mese successivo
  - *es: se abbiamo Gennaio 2012, il mese successivo è Febbraio 2012;*
  - *es: se abbiamo Dicembre 2022,il mese successivo è Gennaio 2013*

```
var mese = 10;
var anno = 2011;
var mese_successivo, anno_successivo;
/* se il mese è dicembre allora il prossimo mese è gennaio e l'anno aumenta di uno */
if(mese==12){
      mese_successivo = 1;
      anno_successivo = anno+1;
}
/* altrimenti l'anno resta lo stesso e il mese aumenta di uno */
else{
      mese_successivo = mese+1;
      anno_successivo = anno;
}
console.log(‘Dato il ’+mese+‘ ’+anno+‘, il mese successivo è ’+mese_successivo+‘
’+anno_successivo);
```

ESEMPIO

- Scrivere un programma che dati un mese (espresso in numeri da 1 a 12) e un anno stampi il mese successivo
  - *es: se abbiamo Gennaio 2012, il mese successivo è Febbraio 2012;*
  - *es: se abbiamo Dicembre 2022,il mese successivo è Gennaio 2013*

```
var mese = 10;
var anno = 2011;
var mese_successivo, anno_successivo;
/* se il mese è dicembre allora il prossimo mese è gennaio e l'anno aumenta di uno */
if(mese==12){
      mese_successivo = 1;
      anno_successivo = anno+1;
}
/* altrimenti l'anno resta lo stesso e il mese aumenta di uno */
else{
      mese_successivo = mese+1;
      anno_successivo = anno;
}
console.log('Dato il '+mese+' '+anno+', il mese successivo è '+mese_successivo+'
'+anno_successivo);
```

Nota: Le parentesi graffe sono necessarie per indicare che ogni ramo è composto da due istruzioni; inoltre, ho aggiunto una tabulazione all'inizio delle righe dei due rami per rendere più chiaro sono blocchi interni

INDENTAZIONE

- Indentare il codice significa utilizzare degli spazi o tabulazioni per distinguere gerarchicamente le diverse parti del codice
- In alcuni linguaggi (es. Python) l'indentazione ha un significato ben preciso, nel senso che guida l'esecuzione
- In Javascript/Java per Processing, non viene codificato dall'interprete in nessun modo, ma ha come scopo solo quello di migliorare la leggibilità e la comprensibilità del codice

ESEMPIO

- Scrivere un programma che stampi la stringa "fa caldo" se la temperatura è superiore a 25 gradi; "fa freddo" se la temperatura è inferiore a 15 gradi; "si sta bene" altrimenti

```
console.log("Esercizio temperatura");
var temperatura = 30;
if (temperatura>25)
    console.log("fa caldo");
else  // temperatura sotto 25 gradi
   if (temperatura<15)
       console.log("fa freddo");
   else
       console.log("si sta bene");
```

L'indentazione del secondo if rende più chiaro che siamo all'interno del ramo else del primo if

AMBIGUITÀ DEGLI IF ANNIDATI

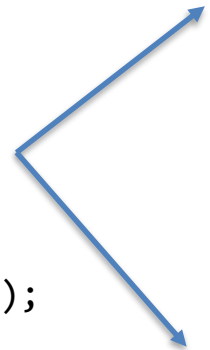
- Es. come viene interpretato il seguente programma?

```
var a=-1;
var b=2;
if(a>=0)
if(b>=0)
console.log("prima stampa");
else
console.log("seconda stampa");
```

AMBIGUITÀ DEGLI IF ANNIDATI (2)

- Es. come viene interpretato il seguente programma?

```
var a=-1;
var b=2;
if(a>=0)
if(b>=0)
console.log("prima stampa");
else
console.log("seconda stampa");
```



```
var a=-1;
var b=2;
if(a>=0)
     if(b>=0)
          console.log("prima stampa");
else
     console.log("seconda stampa");
```

```
var a=-1;
var b=2;
if(a>=0)
     if(b>=0)
          console.log("prima stampa");
     else
          console.log("seconda stampa");
```

AMBIGUITÀ DEGLI IF ANNIDATI (3)

- Ogni else fa sempre riferimento all'if più vicino
- Affinché un else corrisponda ad un altro if occorre usare i blocchi
- Se si scrivono if annidati, il consiglio è di mettere le parentesi graffe per rendere esplicito a quale if ci stiamo riferendoif(a >=0){

```
if(a >=0){
    if(b >= 0)
        console.log("b positivo");
}
else
    console.log("a negativo");
```